computers

MDPI

Article

Multitask Learning-Based Pipeline-Parallel Computation Offloading Architecture for Deep Face Analysis

Faris S. Alghareb [1,*] and Balqees Talal Hasan [2]

[1] Department of Computer and Informatics Engineering, College of Electronics, Ninevah University, Mosul 41002, Iraq
[2] Department of Computer Networks and the Internet, College of Information Technology, Ninevah University, Mosul 41002, Iraq; balqees.hasan@uoninevah.edu.iq
* Correspondence: faris.alghareb@uoninevah.edu.iq

Abstract: Deep Neural Networks (DNNs) have been widely adopted in several advanced artificial intelligence applications due to their competitive accuracy to the human brain. Nevertheless, the superior accuracy of a DNN is achieved at the expense of intensive computations and storage complexity, requiring custom expandable hardware, i.e., graphics processing units (GPUs). Interestingly, leveraging the synergy of parallelism and edge computing can significantly improve CPU-based hardware platforms. Therefore, this manuscript explores levels of parallelism techniques along with edge computation offloading to develop an innovative hardware platform that improves the efficacy of deep learning computing architectures. Furthermore, the multitask learning (MTL) approach is employed to construct a parallel multi-task classification network. These tasks include face detection and recognition, age estimation, gender recognition, smile detection, and hair color and style classification. Additionally, both pipeline and parallel processing techniques are utilized to expedite complicated computations, boosting the overall performance of the presented deep face analysis architecture. A computation offloading approach, on the other hand, is leveraged to distribute computation-intensive tasks to the server edge, whereas lightweight computations are offloaded to edge devices, i.e., Raspberry Pi 4. To train the proposed deep face analysis network architecture, two custom datasets (HDDB and FRAED) were created for head detection and face-age recognition. Extensive experimental results demonstrate the efficacy of the proposed pipeline-parallel architecture in terms of execution time. It requires 8.2 s to provide detailed face detection and analysis for an individual and 23.59 s for an inference containing 10 individuals. Moreover, a speedup of 62.48% is achieved compared to the sequential-based edge computing architecture. Meanwhile, 25.96% speed performance acceleration is realized when implementing the proposed pipeline-parallel architecture only on the server edge compared to the sever sequential implementation. Considering classification efficiency, the proposed classification modules achieve an accuracy of 88.55% for hair color and style classification and a remarkable prediction outcome of 100% for face recognition and age estimation. To summarize, the proposed approach can assist in reducing the required execution time and memory capacity by processing all facial tasks simultaneously on a single deep neural network rather than building a CNN model for each task. Therefore, the presented pipeline-parallel architecture can be a cost-effective framework for real-time computer vision applications implemented on resource-limited devices.

Keywords: computation offloading; deep learning; CNN; edge computing; face recognition; parallelism; pipelining; multithreading; multitask learning; SIMD and MIMD architecture; VGG-Face; YOLOv8

Academic Editor: Paolo Bellavista

Received: 26 November 2024
Revised: 6 January 2025
Accepted: 17 January 2025
Published: 20 January 2025

Citation: Alghareb, F.S.; Hasan, B.T. Multitask Learning-Based Pipeline-Parallel Computation Offloading Architecture for Deep Face Analysis. *Computers* **2025**, *14*, 29. https://doi.org/10.3390/computers14010029

1. Introduction

Face detection and analysis techniques are being targeted as a biometric verification tool to address challenging issues related to computer vision and security such as face recognition, identity verification, human–computer interactions, etc. [1,2]. Face recognition technology (FRT) has garnered significant attention and has been the subject of extensive research and development since the 1990s [3–5]. The concept of face recognition technology refers to a model's ability to distinguish a human face in an image or video stream [6]. Face recognition involves two primary tasks: face detection, which entails finding and locating human faces, and face verification, which relies on recognizing an individual facial features to determine and confirm the identification.

Researchers initially targeted modeling FRT using concepts such as the Markov model and genetic algorithms. However, since AlexNet [7] emerged as the winner of the ImageNet competition in 2012, the trend for modeling FRT has shifted towards using deep learning. Deep Convolutional Neural Networks (CNNs) have been widely adopted to provide intelligent solutions for complex programming scenarios since they can learn new pattern representations and make accurate decisions [8,9]. The use of CNNs to identify a person's facial features has yielded impressive results [2]. For example, the VGG group at the University of Oxford introduced VGGFace [10] for facial features analysis. Subsequently, numerous deep neural networks (DNNs) for face detection and recognition have been developed in the field of deep learning, such as FaceNet, DeepFace, VGGFace and VGGFace2, OpenFace, ResNet Face, etc. [11,12]. Among these, VGGFace has shown excellent results in face recognition for various applications. This can be attributed to its good training using millions of diverse facial images [10]. Furthermore, implementing a real-time face recognition model, i.e., VGGFace, requires an object detector to locate targeted objects and surround them within a bounding box [13]. Several improvements of CNN-based improvements with remarkable detection performance were introduced as object detectors, such as region-based CNNs (R-CNNs), fast R-CNN, and single-shot multibox detector (SSD) [14,15]. However, the YOLO (you look only once) algorithm has become a leading model for object detection due to its ability to accurately detect various objects in real-time [13]. Building on the improvements of earlier YOLO versions, YOLOv8 introduces new features and optimizations, such as incorporating an anchor-free split, to make it an excellent detector for a variety of object detection tasks across numerous applications. Additionally, it offers an optimal accuracy–speed tradeoff while delivering cutting-edge object detection performance [16].

Additionally, tasks related to human faces, such as age estimation, hair detection and classification, gender recognition, and smile detection, have been conveyed in a variety of advanced artificial intelligence (AI) applications such as gesture recognition, pedestrian face tracking, forensic investigations, automated authentication, etc., to provide more accurate descriptions and conformations [4,5]. These facial-based tasks have typically been explored individually using CNNs. However, since tasks based on facial features share correlated attributes, they can be learned simultaneously to significantly reduce training and inference time [2]. Single-task learning (STL) focuses on optimizing models for individual independent tasks. Each model is trained separately for its specific objective, which allows the model to specialize effectively in solving particular problems. Thus, it is inefficient for systems requiring solutions to multiple related tasks simultaneously. It is often used for optimizing task-specific objectives to maintain simplicity and specialization due to its suitability to solve problems with abundant labeled data. Thus, unlike single-task learning (STL), which incurs extensive training time, multitask learning (MTL) has emerged to boost learning performance by concurrently learning multiple correlated tasks [2,17]. The multitasking pipeline aligns more closely with the principles of MTL. In

this context, MTL improves the computational efficiency of the computing architecture by enabling simultaneous face detection and analysis tasks, resulting in improved performance and further generalization of the classifiers. However, the common challenge facing MTL is that learning multiple tasks simultaneously may cause interference and increased model complexity. Interestingly, this can be addressed by balancing classes in the dataset, hyperparameter tuning, and assigning adaptive weights to the loss functions [18].

Several scholarly studies have demonstrated that the MTL approach has improved the performance of individual tasks while also reducing training time, inference time, and computational cost [17–20]. This is due to MTL involving the simultaneous learning of multiple related tasks by sharing features and parameters across complementary classification modules. For instance, training a classifier to learn the gender and age of an individual, rather than training them separately in a sequential manner, can be accomplished in parallel as long as both attributes contribute the same extracted face feature map [1,2].

On the other hand, researchers are intensively concentrating on reducing the need for GPU clusters for AI inference by deploying computationally intensive operations of complicated models with a high-speed, scalable, and cost-effective architecture on CPU-based computing devices [21]. This is due to the accelerated computational ability of a DNN architecture, which can increase its capability to perform computeationally intensive applications with a relatively short execution time, allowing for further human–machine interactions in real-time such as face detection and recognition. Moreover, significant effort has been devoted to enhancing the sophistication of DL models, particularly those deployed on resource-constrained devices [22]. Consequently, edge computing (EC) has proven to be an efficient computing architecture for distributing DL tasks across edge devices and expediting the execution via parallel processing [23]. However, the complexity of DL tasks presents a major burden that hinders the ability to offload computationally intensive models on resource-constrained edge devices. Hence, computation offloading has emerged as an equivalent solution that offloads intensive computations to a powerful processing unit, i.e., a CNN-based feature extraction phase. In contrast, lightweight tasks for decision-making are distributed to edge computing devices [22].

Moreover, partitioning a multi-layer DNN network between on-device inference and edge nodes has been introduced to speed up inference time [8]. In this case, the edge inference handles the initial layers and then uploads the intermediate result to the edge server to process the remaining layers and produce the final result. Nevertheless, when uploading an extensive task to the edge server, it requires a long processing time, impacting overall system performance [24,25]. Therefore, partitioning a DNN model among multiple edge computing devices allows for collaboration on DNN inference between edge devices and the edge server to improve DNN inference performance [25].

A method of distributing DNN inference to an edge device and then dividing the task into multiple threads was presented in [23] to enable parallel processing on a multicore edge processor to further improve processing performance. Despite mainstream adoption of the multicore architecture, there is a lack of studies exploring the use of parallel multicore architectures to speed up the training and, more importantly, inference time for DNN architectures. Therefore, in this study, we propose a novel architecture that offers an innovative approach to optimize the training of deep learning models via the use of MTL and accelerate the inference time on CPU-based resource-constrained edge devices. The proposed architecture employs pipelining and parallelism techniques among the multicores of an edge server and lightweight edge computing devices to efficiently tackle the complexity of the required operations, enabling them to overcome the expense of powerful CPUs. Our proposed parallel-multithreaded model is a fully pipelined neural network architecture that is mainly designed to accelerate the inference of deep face

analysis without compromising accuracy. We leverage multitask learning, pipeline, and multithreading techniques with edge computing to optimize the training and inference time using multiple cores of processors. The proposed architecture involves multiple dependent DNN models incorporated to perform consecutive tasks for deep face analysis. Below, we list the primary contributions of this manuscript as follows:

1. This paper is the first to utilize multithreading to distribute face detection and analysis algorithms across multiple processing cores while exploiting pipelining and parallelism techniques.
2. We combine deep neural networks, computer architecture, and edge computing as complementary fields to develop an innovative computing offloading pipeline-parallel architecture for deep face analysis.
3. The presented model distributes and classifies face analysis tasks (algorithms) based on their level of computational complexity such that intensive computational tasks are assigned to powerful processor cores, whereas lightweight tasks are offloaded to resource-constrained edge computing devices located at the edges of the proposed framework.
4. We implement a unified multitask learning approach throughout the training phase, which allows the model to learn multiple correlated tasks simultaneously, thereby significantly reducing training and inference time and memory usage.
5. We created two custom datasets for head detection and face-age recognition. These datasets were essential for conducting parallel multiple-task training using the MTL approach.
6. We develop a framework architecture that employs edge computing to perform in-depth analyses of facial features. The framework can be implemented in forensic intelligence or adapted for use in human–computer interactive (HCI) applications such as measuring the concentration levels of students in educational systems.

The remainder of this paper is structured as follows. In Section 2, we review a wide range of relevant previously presented approaches that have been developed for face analysis applications such as face detection and recognition, age estimation, hair color and hairstyle classification, gender recognition, and face smile classification. Additionally, we point out how various techniques of computer science and engineering can be combined to deliver improved performance for deep face analysis. Section 3 covers dataset creation and preparation for our proposed face analysis model. The proposed pipeline-parallel architecture for deep learning face analysis is introduced and discussed in Section 4. Section 5 explains the experimental setup steps and deployment process of the required hardware equipment and software IDE environments. The results are elaborated on and deeply analyzed in Section 6. Finally, Section 7 summarizes the manuscript and suggests potential applications for deployment.

2. Related Work

In this section, we provide a comprehensive and up-to-date review of several deep learning networks for human face detection and verification, edge computing, and multitask learning that have been recently published in the literature. The selected prior works span different levels of abstractions, including deep learning [26], face recognition [3,27], hair analysis [5,28], edge computing and computation offloading [29–32], parallelism approaches [30,33], multithreading and multi-processes [23], and multitask learning [1,2,34] to capture feature maps of faces and perform different facial analysis tasks such as face detection and verification, landmark localization, smile detection, gender recognition, age estimation, hair color and style classification, etc. Our aim is to provide a comprehensive review that is valuable for designers of cost-effective computing paradigms for face anal-

ysis based on deep neural network architectures in order to achieve high-performance computing architectures based on edge computing.

Deep CNNs have been extensively utilized in detection and recognition tasks [35]. For instance, age estimation based on the VGGFace model was conducted in [26] to estimate the age of individuals based on their facial features. Deep face recognition datasets including VGGFace [10] and VGGFace2 [36] were introduced by the VGG group to train several CNNs and deliver remarkable facial recognition accuracies. In a recent study by Hasan et al. [3], an improved model for facial recognition is presented by training ResNet-50 using the VGGFace architecture, presented in [10]. The proposed model addressed the issue of training a DL model with a limited number of facial images per individual. To increase its recognition performance, the authors re-trained a pre-trained facial recognition model with a different dataset. Combining VGGFace2 for facial feature extraction with PCA was explored in [27] to reduce Gabor features, increasing recognition accuracy.

Additionally, deep learning has been widely used for hair analysis and classification across various helpful applications. For instance, human appearance can be categorized based on hair. Additionally, hair can be a crucial feature for recognizing familiar faces [37], and more accurate gender recognition can be achieved via hair analysis [38]. Moreover, hair color classification can be a beneficial toolkit for forensic intelligence [39]. Therefore, DL-based hair recognition and classification have been explored across a wide range of applications. Hair detection and hairstyle classification was examined in [28]. The authors were the first to attempt to perform hair detection and hairstyle classification to classify seven different classes. A principal component analysis (PCA) was employed to shrink the extracted feature maps into a flattened vector containing 4096 pixels. A CNN based on the CaffeNet-fc7 model with random forest (RF) provided the best-obtained hair features at the patch level, while local ternary patterns (LTP) for features with a support vector machine (SVM) for decision-making realized the best segmentation results at the pixel level. A systematic review was conducted in [5] recently regarding hair and skin analysis. The review confirms that machine learning approaches such as artificial neural networks (ANNs) and SVMs achieved the highest prediction accuracy (95% and 90%, respectively).

A fully convolutional multi-task neural network was introduced in [40] to extract external facial image features (baldness and hair color). The proposed framework achieved an accuracy of 92% and 93% for non-hair and average hair color prediction, respectively, using the random forest classifier. Likewise, a novel diffusion-based generative model was presented in [41] to segment human hair and non-hair from the background of an image. The model accuracy was validated on three datasets (Figaro-1k, CeleA Mask-HQ, and Face Synthetics) and realized accurate hair segment predictions of 98.03%, 95.34%, and 98.79% for the tested datasets, respectively. However, these hair analysis models performed the tasks sequentially without using overlapping methods, such as in a pipeline, and they did not implement parallel processing to exploit multitask learning (MTL) simultaneously.

On the other hand, intrinsic attributes such as age, gender, and smile recognition have been examined in various face images. The authors in [42] examined the relationships between age, head pose estimation, and gender recognition. Similarly, an MTL-based facial analytics framework was introduced in [43], which integrates gender recognition and age estimation with other facial expression analyses. The hybrid learning approach for smile recognition (HLSR) was introduced in [44]. The authors fused a CNN with the XGBoost algorithm for real-time facial smile expression recognition.

Parallel module processing was proposed in [45] for medical diagnosis to distinguish between depressed and normal patients based on facial expression analysis. The proposed Dep-FER model is capable of distinguishing facial expressions including fear, anger, sadness, happiness, surprise, and more using three parallel modules. These module were

mainly designed to identify samples, learn general similarities, and predict an intrinsic relationship between normal and depressed patients. Likewise, considering parallel computing paradigms, the study conducted by Gamatie et al. [33] was the first to explore the integration of lightweight, low-power cores dedicated to parallel processing with high-speed processor cores designed for sequential execution. The authors investigated energy efficiency by implementing multicore architectures on resource-constrained edge computing devices, aiming to establish a cost-effective computing paradigm. Later on, energy efficiency with AI edge computing was explored in a study by Cheikh et al. [30], which employed an interleaved multithreading approach to distribute heavy computations across multiple edge computing devices. The study demonstrated the efficacy of the presented approach by exploiting the synergy between thread-level parallelism (TLP) and data-level parallelism (DLP). It was confirmed that leveraging TLP and DLP to configure a multiple-instruction multiple-data (MIMD) architecture achieves higher speedup for more complicated 2D convolutions compared to a single-instruction multiple-data (SIMD) architecture.

On the other hand, situating computationally intensive DNN tasks on mobile edge computing devices for real-time DNN inference execution was first introduced in [8]. The authors proposed the neurosurgeon approach, referred to as the DNN partitioning method, which partitions the computations of a DNN between the cloud and a mobile device. It achieves significant performance improvements such as reduced latency and energy consumption. To further improve inference on edge computing, an optimized deep learning model was presented to classify vehicle images and whether they were stolen [29]. Fine-tuning the tested model reduced the model's size, increased the model's accuracy, and slightly improved the model loading time on edge devices. Likewise, the authors in [46] investigated the tradeoff between computing accuracy and latency of computationally intensive deep learning applications deployed on edge devices with limited resources. Moreover, in [47], a collaborative approach for thread-offloading was proposed to reduce the energy consumption of multithreading by provisioning optimized TLP on mobile edge computing (MEC). Similarly, the TLP was leveraged on multiple edge devices to shorten the inference time and maximize cloudlet throughput [48]. Notably, prior research related to enabling DNN inference on the edge can be divided into three categories: cloud-assisted inference, edge-assisted inference, and multi-device collaboration [49,50]. Herein, we concentrate on reviewing offloading tasks on multiple edge devices. A distributed DNN computing architecture known as CoEdge was presented in [50], in which inference tasks were divided into feature extraction and target classification. The input image is split into small patches of different sizes and distributed on the edge devices during the feature extraction phase, then the outcomes are aggregated during classification to compute the final execution. Similarly, the DNN inference was divided into two parts deployed on the user equipment (UE) and edge server [32]. The UE performs feature extraction, then compresses the feature map and sends it to the edge server. The edge server then passes it to the remaining DNN part to provide the predicted outcome, which is then sent back to the UE. Additionally, to expedite the processing tasks of DNN models, a parallel-pipeline inference was introduced by Goel et al. [31]. This architecture splits the DNN network into sets of hierarchical consecutive layers, and each set is run on a different device. By balancing the workload between the cooperative devices, the presented approach is capable of concurrently processing multiple video frames, thus achieving improved throughput.

Multitask learning (MTL), on the other hand, has been adopted in several computer vision problems for efficient automation of deep learning applications [1,2,34,51]. Caruana et al. [52] were the first to introduce and generalize the MTL concept. Subsequently, MTL has expanded to enable CNN deployment for detailed facial analysis. A VGG-based unified network architecture called UberNet, expanding the training of multiple tasks on a single

CNN network, was introduced in [51] using diverse datasets. UberNet employs MTL for simultaneous training of CNNs by fusing all intermediate layers [51]. Moreover, a region-based CNN (RCNN) model was proposed in [34] to perform human pose estimation and action recognition. The model achieved an action mean average precision (mAP) of 70.5%. Nevertheless, action classification models are still targeted to improve their classification performance. Additionally, for images containing occlusions with faces, Zhang et al. [1] presented a task-constrained deep learning model that performs multiple correlated tasks of facial features concurrently. The proposed task-constrained deep convolution network (TCDCN) model is utilized for facial landmark detection, gender recognition, and object detection of smile and glasses. In a similar manner, real-time facial analytics based on event surveillance cameras were presented later in [17] to estimate individuals' visual attention such as head pose, eye gaze, and facial occlusions for the analysis of human attention levels.

Additionally, facial feature analysis to recognize and classify multiple face-based tasks was presented in the HyperFace deep CNN model in [2]. The model effectively exploits multitask learning, such as face detection, gender recognition, landmark localization, and head pose estimation simultaneously. The approach utilizes a diffusion method that leverages combinations of rich features among low, intermediate, and high convolutional layers; therefore, various extracted facial features are fused from different layers of the network. The authors first used a truncated AlexNet [7] as a backbone, which consists of five convolutional layers for evolving feature maps. These fused features are then fed to five ANN classifiers that operate in parallel to perform different face tasks. Following that, the ResNet-101 [53] model was tested with a fiducial extractor to investigate the fusion of intermediate layers with MTL, which was found to deliver more accurate prediction performance for all tasks. Using HyperFace for the AlexNet-based model, accuracies of 97% and 94% were achieved for gender recognition tested on CelebA and LFWA, respectively, whereas 98% and 94% gender prediction outcomes were realized using HyperFace ResNet as the backbone, validated on the same datasets. Moreover, using the IBUG dataset, ResNet-101 was discovered to deliver the highest landmark localization (91.82%) for 68-point. However, this achievement was realized at a high layer density, which incurs larger storage capacity costs and requires massive, computationally intensive operations. Likewise, several facial-related tasks such as face detection and recognition, age and gender estimation, and emotion prediction were addressed by a single framework in [23]. However, this study was mainly conducted to compare employing multithreading versus multi-processes on an edge computing device for low-cost deep learning applications. The experimental results showed that when the hardware resources—i.e., the number of CPU cores and memory cells—of the edge device are limited, leveraging multithreading improves performance compared to deploying multi-processes architecture. Moreover, running two threads in parallel on two modules, each processing an image, results in an 18.87% speedup when using an NVIDIA Jetson TX2 device and a 19.83% speedup when using an AGX edge computing device. Nevertheless, employing MTL approaches for parallel facial feature analysis to detect faces, determine identity, recognize gender, and estimate age has not yet been adequately addressed. In other words, efficient employment of MTL with face analysis requires more in-depth analysis of human faces.

To summarize, these relevant prior DL networks and computing architectures offer individual benefits such as accelerated training, deployment of DL models on resource-constrained computing devices, and parallel implementation. They have confirmed the employment of a single or dual-combined specific technique, but there is no attempt to combine desirable techniques of computer science and engineering such as parallelism, pipelining, computation offloading, multitask learning, and multithreading in a single framework architecture to provide deep face analysis. Considering these aforementioned

challenges, in this paper, we leverage DLP and TLP, pipelining, computation offloading, and multitask learning with edge computing paradigms to develop a pipeline-parallel architecture for deep face analysis. Primarily, our paper is the first to integrate the pipeline technique between the edge server and edge devices to maximize the throughput of DNN-based tasks. The innovative pipeline-parallel with a computation offloading architecture delivers a thorough facial analysis of the captured faces in an image taken in the wild. It can detect multiple faces in a scene, crop all detected heads surrounded by bounding boxes, and create their corresponding face images. These images are then saved in the database on the hard disk, which is then processed through the pipeline-parallel architecture for feature extraction using the VGGFace network. This involves feeding the extracted feature maps into three MTL-based classifiers (hair-module, gender-module, and faceID-module) designed for deep face analysis including hair color and style prediction, gender and smile recognition, and face identification and age estimation, respectively. Notably, these classifiers operate in parallel based on a multithreading approach that distributes tasks among seven processing units (cores); four cores are for the server edge, while the remaining three are for the edge devices to achieve a low-cost and efficient computing architecture.

3. Dataset Creation and Description

In this section, three categories of datasets with distinct attributes are utilized for training and testing the proposed deep face analysis architecture, as listed in Table 1. The datasets were used to train the object detection and cropping model (YOLOv8) and three classification modules (hair, gender, and faceID module) working in parallel to provide MTL simultaneously. Next, we discuss the creation and preparation of the used datasets.

Table 1. Summary of the utilized datasets.

Dataset	Task	Encoded Vector Length	Total # of Samples	# of Train Images	# of Valid. Images	# of Test Images
HDDB	Head Detection	5-label	2706	1894	406	406
CelebA	Gender and Smile recognition	5-bit	20,000	16,000	2000	2000
CelebA	Hair color and style prediction	7-bit	20,000	16,000	2000	2000
FRAED	Face recognition and Age estimation	32-bit	1400	1120	140	140

3.1. Head Detection Dataset

Face recognition technology has been embedded in several biometric-based access and control devices such as smartphones, attendance verification devices, entrance of smart buildings, etc. [6,54]. Such applications and technologies mainly function based on deep neural networks (DNNs). DNNs have been trained and validated based on numerous standard face detection datasets such as Annotated Face in the Wild (AFW) [55], Disguise Faces in Wild (DFW) [56], and Face Detection Dataset and Benchmark (FDDB) [57]. However, despite the availability of several face detection datasets, there is still a need for datasets containing annotations not only for faces but also for heads. This is due to existing datasets concentrating merely on facial features, leading to much valuable information about the head being ignored. For instance, further details about a person's identity description can be their hair color and style. Consequently, a new dataset for head detection and analysis is introduced herein, namely Head Detection Dataset and Benchmark (HDDB).

The created dataset facilitates training DL-based models for face detection and ID verification to detect the entire head, including the hair color and style, which in turn

can provide a deeper analysis of individuals and achieve accurate descriptions. Thus, the created HDDB dataset is the first dataset that explicitly annotates the borders of humans' heads rather than only their faces. HDDB has 4932 head annotations of 2706 images. We randomly selected 2706 images from the FDDB dataset. Then, using the VGG Image Annotator (VIA) tool [58], a bounding box around each existing head was drawn for all selected images. These 2706 images contain 4932 annotated heads, and they were used to train the nano-based YOLOv8 head detection model. The annotations are stored as CSV files in a COCO format. Thus, after retrieving the dimensions of each image, we added two new columns ('img_w' and 'img_h') to the image annotation CSV file. Afterward, we transformed the head annotation information from the COCO format, which uses top-left corner coordinates ('x' and 'y'), width ('w'), and height ('h') values for bounding boxes, to the YOLO format, which represents bounding boxes with the center coordinates ('center_x' and 'center_y') and dimensions ('width' and 'height') normalized according to the image's size. The transformation process is further discussed in Algorithm 1. The final dataset is organized into two separate directories, one for images and the other for their annotations. Lastly, the dataset is split into subsets for training, validation, and testing. Initializing the data (.yaml) file is essential for training the YOLOv8 model. Figure 1 depicts the concept of the head detection bounding box versus face detection. As observed, the modified YOLOv8 model crops the heads to capture detailed features of individuals, including their hair.

Figure 1. Head detection dataset versus face detection dataset using nano-based YOLOv8.

Algorithm 1 HDDB Dataset Creation

Input : FDDB Dataset
Output : New Head Detection Dataset (HDDB) with data.yaml file
1: **procedure** ANNOTATE_HEADS()
2: **for** each image in HDDB_Images_Directory **do**
3: Manually annotate head bounding boxes in the image using the VIA tool
4: *COCO_CSV* ← Save COCO-format head bounding boxes to CSV in COCO_Directory.
5: **end for**
6: **end procedure**

Algorithm 1 *Cont.*

procedure CONVERT_COCO_TO_YOLO()
  for each image in HDDB_Images_Directory **do**
    $yolo\_annotation \leftarrow$ NULL
    Get the retrieved image dimensions:
      $image\_width = image['width'], image\_height = image['height']$
    Get the image's COCO annotation CSV from COCO_Directory.
    for each annotation in the image's COCO annotations file **do**
      Convert the COCO bounding box coordinates to YOLO format:

$$yolo_x = \frac{coco['X'] + coco['W']/2}{\text{image\_width}}, \qquad yolo_y = \frac{coco['Y'] + coco['H']/2}{\text{image\_height}}$$

$$yolo_w = \frac{coco['W']}{\text{image\_width}}, \qquad yolo_h = \frac{coco['H']}{\text{image\_height}}$$

      Append YOLO-formatted annotation to the YOLO annotation string:
        $yolo\_annotation +=$ `f"`{annotation['category_id']} {yolo_x} {yolo_y} {yolo_w} {yolo_h} $\backslash n$`"`
    end for
    Create YOLO annotation.txt after the image in Yolo_Directory.
  end for
end procedure
procedure MAIN()
    $HDDB\_Images\_Directory \leftarrow$ Randomly Select 2706 images from FDDB and add to HDDB_Images_Directory.
    $COCO\_Directory, Yolo\_Directory \leftarrow$ Create directories for COCO and YOLO CSV files
    Execute Annotate_Heads(),Convert_COCO_to_YOLO() Procedures
    Create data.yaml, essential for training the YOLOv8 model
end procedure

3.2. Gender-Smile Dataset

To train and test the proposed gender-smile module, a balanced subset of data samples from the CelebA dataset [59] was selected, with each class containing about 10,000 image samples. CelebA consists of more than 200,000 celebrity images. Each image is annotated with 40 binary facial attributes. To deliver accurate gender and smiling recognition of an individual in a variety of head pose situations, the gender-smile module requires a well-trained classifier. Thus, careful consideration was given to selecting a diverse range of images that capture individuals' faces from multiple viewpoints, including frontal and far views, in both smiling and non-smiling scenarios. Each image in the prepared gender-smile dataset is annotated with five labels, including ["image_id", "male", "female", "smiling", and "non-smiling"]. Moreover, one-hot encoding is utilized to represent each attribute, where presence is encoded as '1' and absence as '0'. Therefore, each image has two separate one-hot vectors to accurately reflect the binary nature of the "gender" and "smile" attributes, as can be seen in Algorithm 2 (lines 9 to 20).

Algorithm 2 Gender-Smile Dataset Preparation

Input : CelebA Dataset
Output : Gender-Smile Dataset
procedure GENDER_SMILE_DATASET_CREATION()
    Temp_Dataset_Directory ← Select an equal number of images from the CelebA dataset, divided into four categories: male smiling, male not smiling, female smiling, and female not smiling.
    Gender_Smile_Directory ← Create directory to store images of Gender_Smile Dataset
    Gender_Smile.CSV ← Create CSV file to save Gender_Smile Dataset Annotations.
  for each each image in Temp_Dataset_Directory **do**
    Add image to Gender_Smile_Directory
    image_id ← image_name
    Extract binary attributes: gender, smile
   Initialize gender_vector
    if (gender == "male") **then**
      gender_vector = [1, 0]
    else
      gender_vector = [0, 1]
    end if
   Initialize smile_vector
    if (smiling) **then**
      smile_vector = [1, 0]
    else
      smile_vector = [0, 1]
    end if
    Create one_hot_encoding = [image_id, gender_vector, smile_vector]
    Append one_hot_encoding as a new row to the Gender_Smile.csv
  end for
end procedure

3.3. Hair Color-Style Dataset

Here, the hair color-style module was trained and tested using a subset of data samples that were also chosen from the CelebA [59] dataset. A careful selection of images was made from the CelebA dataset, similar to the gender-smile dataset, to create a subset that includes various hair colors and styles. Each class contains between 5000 and 7000 image samples, except for the gray hair category, which has approximately 700 image samples due to the limited number of images available for that color in the original CelebA dataset. However, the primary focus of this manuscript is to demonstrate the architectural efficiency of the pipeline-parallel design rather than to fine-tune model learning loss strategies, which have been well-studied in the literature. Each image in the hair color-style dataset is annotated with seven labels, which are ["image_id", "black-hair", "blond-hair", "brown-hair", "gray-hair", "wavy-hair", and "straight-hair"]. Likewise, to encode the binary representation for the attributes of hair color and style, two distinct one-hot vectors were configured for each image, in which presence was encoded as '1' and absence as '0' using one-hot encoding. These steps are deeply explained in Algorithm 3. In Figure 2, we show groups of the hair color and style dataset used to train the hair module of the proposed deep face analysis architecture.

Figure 2. Sample images from the hair dataset used to train the hair color-style module.

Algorithm 3 Hair Color-Style Dataset Preparation

Input : CelebA Dataset
Output : Hair Color-Style Dataset

procedure HAIR_COLOR_STYLE_DATASET_CREATION()
    Temp_Dataset_Directory ← Select a subset of images from the CelebA dataset that includes hair color and style attributes.
    Hair_Color_Style_Directory ← Create directory to store images of Hair_Color_Style Dataset
    Hair_Color_Style.CSV ← Create CSV file to save Hair_Color_Style Dataset Annotations.
  for each each image in Temp_Dataset_Directory **do**
    Add image to Hair_Color_Style_Directory
    image_id ← image_name
    Extract binary attributes: color, style
   Initialize color_vector
    if (hair_color == "black") **then**
      hair_color_vector = [1, 0, 0, 0]
    else if (hair_color == "blond") **then**
      hair_color_vector = [0, 1, 0, 0]
    else if (hair_color == "brown") **then**
      hair_color_vector = [0, 0, 1, 0]
    else
      hair_color_vector = [0, 0, 0, 1]
    end if
   Initialize style_vector
    if (style == "wavy") **then**
      hairstyle_vector = [1, 0]
    else
      hairstyle_vector = [0, 1]
    end if
    Create one_hot_encoding = [image_id, color_vector, style_vector]
    Append one_hot_encoding as a new row to the Hair_Color_Style.CSV
  end for
end procedure

3.4. Face Recognition and Age Estimation Dataset

To create a multitask dataset for face recognition and age estimation, we initially categorized individuals into four groups representing people in their 10s, 30s, 50s, and 70s. Within each group, we carefully selected seven famous people of a specified age to serve as references for both facial recognition and age estimation tasks. Figure 3 depicts selected image samples from the created face recognition and age estimation dataset (FRAED). As observed, the individuals were divided into four age groups. The first group consisted of people under 10, and the last group included individuals in their 70s.

Furthermore, Google Images and Pinterest were used to collect 50 images for each person. The Shotwell application, which can be reached on Ubuntu 23.10, was then used to crop the region of interest (head) from each image. The created database consists of 28 individuals grouped into 4 subsets, with each subset containing 7 people. In total, we collected 1400 images for face recognition and age estimation, as listed in the last row of Table 1. After that, each image in the created face-age dataset was annotated within 32 labels, including ["image_id", individuals_names, 10's, 30's, 50's, 70's]. Moreover, one-hot encoding was utilized to represent each attribute, where '1' represents presence and '0' represents absence. Since there were two tasks, i.e., face recognition for 28 people and age estimation based on 4 age groups, 2 separate vectors were combined and encoded for image annotation. Thus, each annotation vector was encoded as 32-bit, containing two active bits to determine the person's identification and the estimated age of each head image. The entire procedure for creating the dataset is further explained in Algorithm 4.

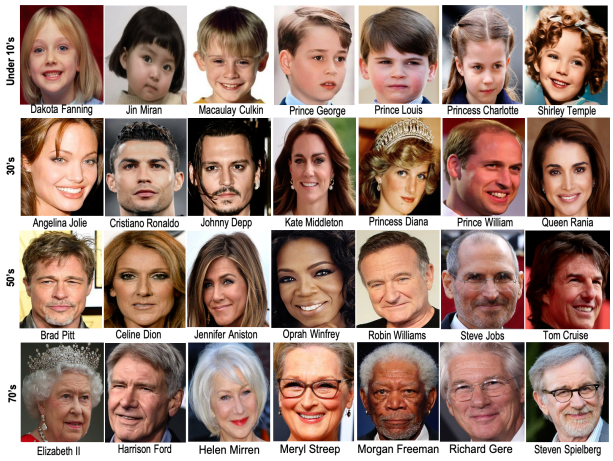

Figure 3. Selected image samples of the created face recognition and age estimation dataset.

Algorithm 4 Face Recognition and Age Estimation Dataset Creation

Input : Google Images and Pinterest are used to collect 50 images for each individual
Output : Face Recognition + Age Estimation Dataset (FRAED)

procedure FACE_RECOGNITION_AGE_ESTIMATION_DATASET_CREATION()
    Initialize the required age groups and the names of famous individuals
    Age_Groups = [10's, 30's, 50's, 70's]
    Famous_People_in_10s = ["Dakota Fanning", "Jin Miran", "Macaulay Culkin", "Prince George", "Prince Louis", "Princess Charlotte", "Shirley Temple"]
    Famous_People_in_30s = ["Angelina Jolie", "Cristiano Ronaldo", "Johnny Depp", "Kate Middleton", "Princess Diana", "Prince William", "Queen Rania"]

Algorithm 4 *Cont.*

```
    Famous_People_in_50s = ["Brad Pitt", "Celine Dion", "Jennifer Aniston", "Oprah Winfrey", "Robin Williams", "Steve Jobs", "Tom Cruise"]
    Famous_People_in_70s = ["Elizabeth II", "Harrison Ford", "Helen Mirren", "Meryl Streep", "Morgan Freeman", "Richard Gere", "Steven Spielberg"]
    No_of_Images_per_person ← 50
    Face_Rec_Age_Directory ← Create directory to store the dataset's images
    Face_Rec_Age.CSV ← Create a CSV file, with 33 columns: one for the image ID, 28 for the famous individuals, and four for the age groups themselves.
  for each age in Age_Groups do
    for each Famous_Person in age do
      for i = 0 to No_of_Images_per_person do
        image ← Download image for Famous_Person from Pinterest and Google Images
        head_Img ← Crop the head from the image using Shotwell tool
        Store head_Img in the Face_Rec_Age_Directory
        Annotate the head_Img:
          Set "image_id" label to the unique ID of the head_Img
          Encode Face_Rec hot-vector: For each famous person, set its label from the face recognitions 28 labels to '1' if present, else '0'.
          Encode Age hot-vector: For each age group, set the label from the 4 Age_Groups to '1' if present, else '0'.
          Create one hot encoding = [image id, Face_Rec vector, Age vector]
          Append one_hot_encoding as a new row to the Face_Rec_Age.CSV
      end for
    end for
  end for
end procedure
```

4. The Developed Parallel MTL-Based Face Analysis Framework

Deep learning applications such as face analysis require billions of intensive computational operations. To run these applications in real-time, computationally efficient and low-latency architectures are desired. Therefore, pipeline and parallel implementation approaches are utilized to achieve high-speed performance for iterative computations. The pipelining technique is primarily employed to overlap the execution of DNN models to improve performance. It is used to achieve two main goals: first, to perform dependent tasks in parallel on different input data, i.e., different input images, and second, to speed up the processing performance by dividing the workload on the available processing cores. The overlap among algorithms is accomplished via multithreading, in which multiple threads (algorithms) are executed in a parallel manner. The proposed architecture performs computer-intensive operations on the server edge (core i5 processor), which employs parallelism and pipeline techniques to deliver a high throughput. Meanwhile, the lightweight tasks are offloaded to edge nodes, i.e., Raspberry Pi 4. Therefore, the input images are processed incrementally through multiple DNN models configured using pipeline and parallel techniques. The general framework of the proposed deep face analysis architecture is depicted in Figure 4.

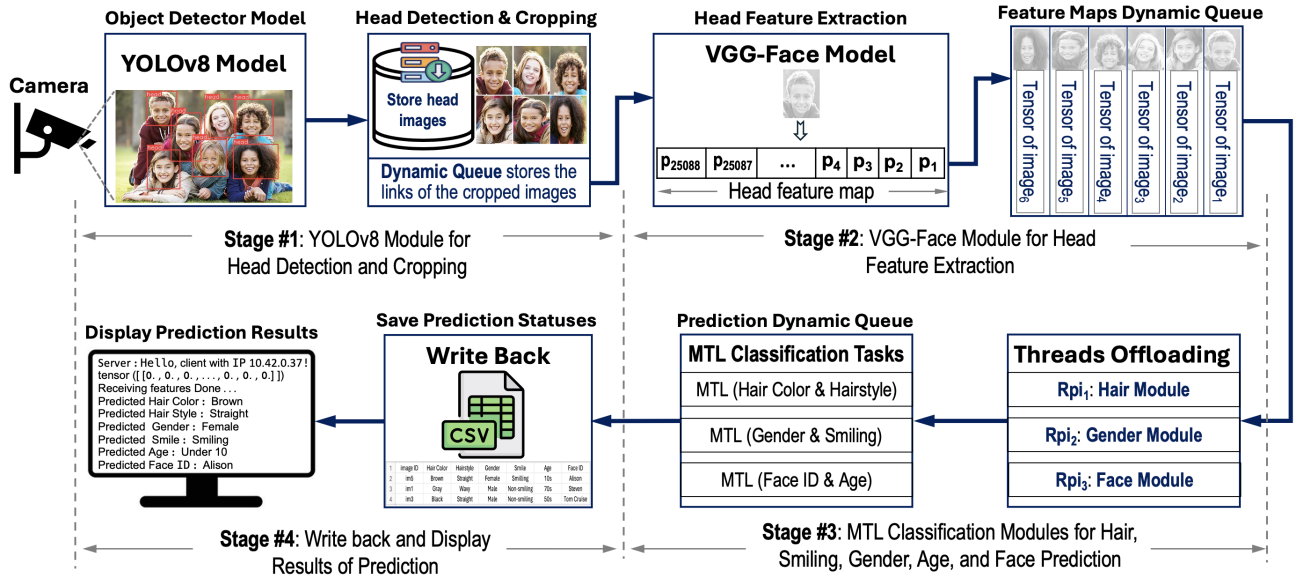

Figure 4. The general framework of the proposed deep face analysis architecture.

The concept of the proposed architecture is similar to the HyperFace analysis network proposed in [2]. However, it differs in several key aspects. First, our developed architecture is based on YOLOv8 from Ultralytics [16] and VGG-Face [10] as backbone structures. These DNN models were implemented using pipeline and parallel techniques to realize high-computing performance. Additionally, we created a task-specific dataset, the HDDB dataset, which is tailored for head detection. Essentially, we annotated the HDDB dataset with the YOLO format for training YOLOv8 to detect heads instead of faces. The detected heads are then cropped and stored on the hard disk of the server computer to be passed through the next stage of the pipeline network for feature extraction. Second, to capture complex features of heads, small kernels need to be utilized in the convolutional layers of the CNN [35]. Therefore, we employ a pre-trained model of VGG-Face to accomplish hair and facial feature extraction since it adopts only 3 × 3 kernels in all convolutional layers. Thus, the input to the VGG-Face network is a head image resized to a resolution of 227 × 227. In the third stage of the pipeline, feature maps are offloaded to edge computers to perform parallel multitask classification. Finally, during stage four, the classification results are sent from the edge device back to the edge server to provide detailed face analysis and decision-making. Therefore, the developed framework is designed to process four images incrementally through four pipeline stages. For instance, when all four stages become full, it indicates that the output decision-making of the first head image is in the write-back stage. The second image is being classified using three parallel MTL-based classification modules. The third image is undergoing feature extraction using VGG-Face. Lastly, YOLOv8 scans the fourth image for head detection in the first stage. These parallel incremental operations are clearly illustrated in Figure 5.

Moreover, a producer–consumer technique, a well-known approach in concurrent programming, is adapted to maintain synchronization and temporary storage sharing between the pipeline stages. The approach is employed to handle two categories of threads: producers and consumers. A producer inserts items into the buffer, whereas a consumer takes items out of the buffer. Furthermore, to achieve efficient shared-memory allocation, a dynamic-size buffer (queue) is utilized in which the buffer size is altered depending on the number of detected individuals (heads) in the input image.

In the proposed pipeline-parallel architecture, we leveraged the producer–consumer approach to facilitate memory sharing. Initially, at stage #1, the YOLO producer thread generates paths of the cropped images and adds them to YOLO queue. During stage

#2, the VGG-thread serves as both a producer and a consumer. It retrieves the paths of the cropped images from the YOLO queue and passes each head image to the VGG-Face model to generate a tensor that represents the extracted feature map for the corresponding individual. In this case, the YOLO queue is shared between the YOLO producer thread and the VGG consumer thread. The generated tensors at the output of the VGG-Face model are then placed into the offloading queue(s). Depending on the number of distributed edge devices, illustrated in Figure 6a,b, two offloading producers were considered. For the architecture involving a multithreaded server with a single Raspberry Pi (Rpi), there exists a single offloading queue where the VGG-thread deposits the feature maps.

On the other hand, in the case of a multithreaded edge server with three Rpi devices, the VGG-thread copies the feature tensors to three offloading queues, each corresponding to an Rpi. We discuss this further in Section 4.3. When an edge device connects to the edge server, the corresponding client thread consumes a feature map from the client's queue and sends it to the respective device. For instance, when Rpi1 sends a request to the server asking to send an image tensor for hair color and style classification, the server accesses the feature map queue of Rpi1, retrieves an image tensor, offloads it to Rpi1, and waits for prediction outcomes. Once Rpi1 finishes predicting hair color and style, it sends the predicted classes back to the server. The server enqueues the received outcomes of hair into the prediction queue. Meanwhile, it obtains another image tensor from the tensor queue of Rpi1 and offloads it to edge node Rpi1. Thus, in stage #3, the server assigns a thread for each edge device request. However, each thread performs two sub-tasks. The first sub-task retrieves a new feature tensor from the feature map queue and offloads it to the client, while the second sub-task enqueues the received predictions into the CSV prediction queue. During stage #4, a write-back thread assigned by the server consumes predictions from the prediction queue and writes the results to a CSV file for saving. Consequently, as depicted in Figure 5, the inference of the proposed architecture is designed to process four images (im1, 2, 3, and 4) of different individuals concurrently. This implies that when the first image (im1) reaches the write-back stage to be saved in a CSV file, the predicted classes of im2 are enqueued into the prediction queues, the tensor of im3 is inserted into the offloading queue, and the path of the cropped head (im4) from the YOLO model is placed into the YOLO queue.

The synchronization in the proposed pipeline-parallel architecture was achieved by incorporating storage units (SUs) and the producer–consumer approach between the pipeline stages. Therefore, the use of these mechanisms effectively addresses variations in processing delays, such as those induced by computationally intensive tasks like YOLOv8 for head detection and cropping or VGG-Face for feature map extraction, ensuring seamless data flow between the stages despite task dependencies or variations in execution time. To the best of our knowledge, the proposed edge computing-based architecture is the first to employ pipeline and multithreading techniques in a single framework for in-depth face analysis of multiple people. In Section 6.3, we analyze the performance evaluation of the proposed pipeline-parallel architecture over the sequential design along with the STL and MTL approach. Next, we further discuss our proposed pipeline-parallel architectures.

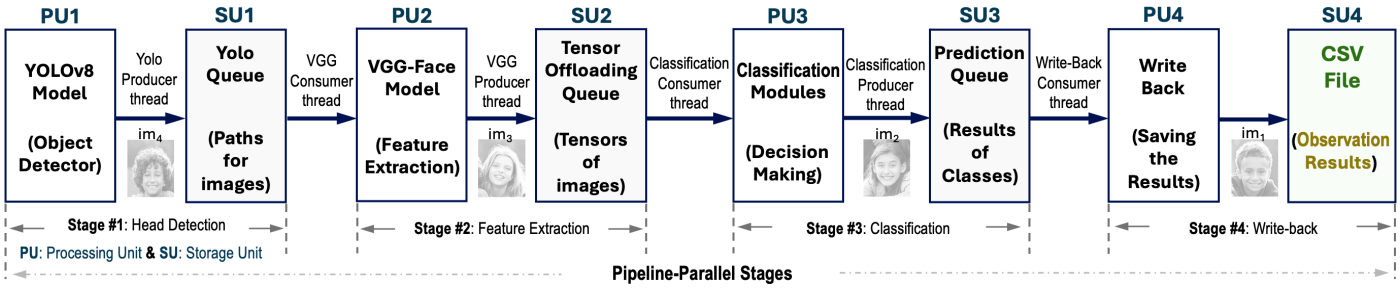

Figure 5. Stages of the pipeline-multithreading architecture, showing four images being processed in parallel.

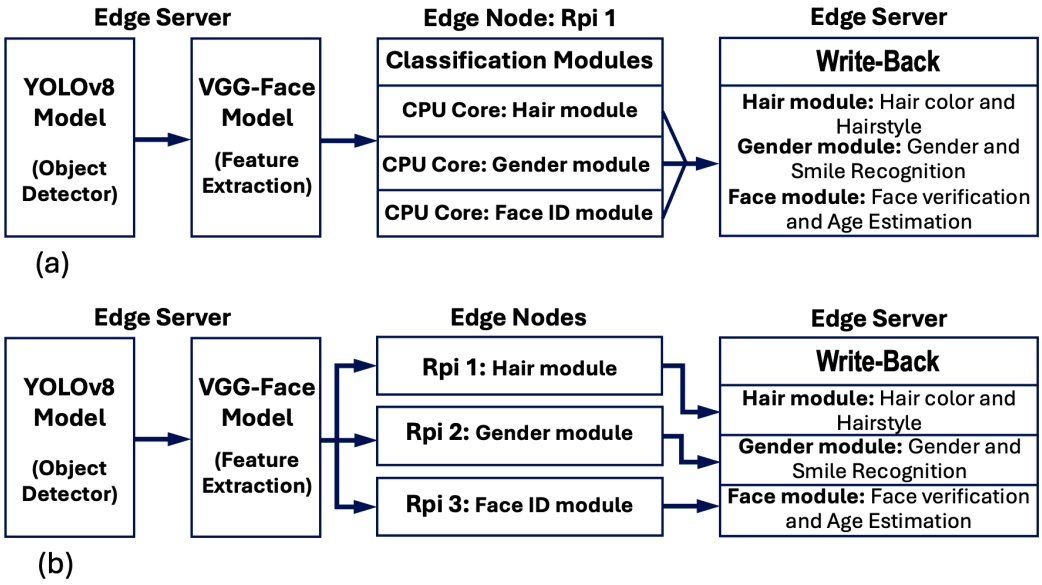

Figure 6. Proposed pipeline-parallel architectures with thread distributions; (**a**) multithreading three MTL-based classifiers on a single edge device, (**b**) multithreading three MTL-based classifiers on a cluster containing three edge computing devices.

4.1. The Proposed Pipeline-Parallel Face Analysis Architectures

In this paper, three DNN architectures for deep face analysis are presented, since the procedure of face analysis architecture is a multithreaded application. The first design is sequential-based and involves cascaded networks. It is considered as the baseline architecture. The second and third architecture designs utilize pipeline and parallel processing techniques with multithreading to expedite the computationally intensive operations of the DNN models. These developed designs are as follows:

- The sequential-based architecture: It deploys a conventional fully sequential design. In this design, the processing is divided into three well-tailored cascaded modules (YOLOv8 module1, VGG-Face module2, and Classification module3), and each module must be finished before the next one can start.
- Pipeline-parallel architecture with multithreading on a cluster of edge device: This employs parallel and pipeline techniques on the edge server and multithreading of the classifiers on a cluster of three homogeneous edge devices, running multiple lightweight processors to perform multiple classification tasks.
- Pipeline-parallel architecture with multithreading on a single edge device: Similar to the previous parallel design, this utilizes multithreading on the server edge between module1 and module2. However, here, all classification tasks are offloaded to a single edge device. Thus, under this design, four threads run on a quad-core processor of the edge device; each core performs a single classification task at a time. Notice that one

of the four cores of the edge device is allocated for the main thread, which handles the threads' organization with other processing cores.

Figure 6 depicts the proposed pipeline-parallel architectures. As seen, Figure 6a illustrates the design of the pipeline-parallel architecture with multithreading on a single edge node, while the design of the pipeline-parallel architecture with multithreading on three edge devices is shown in Figure 6b. These proposed architectures employ pipeline and parallel techniques along with multithreading to realize high-speed performance for deep face analysis.

4.2. Multi-Task Learning

Herein, we propose an expandable pipelined and parallel implementation for deep facial feature analytics. Multitask modules are proposed to identify various facial expressions such as hairstyle, hair color, gender, smile, age, and face verification. We adapt the approach of learning multiple related face analysis tasks to train two complementary classifiers using a single MTL module simultaneously. This is because the feature maps extracted from an image are correlated among the proposed classifiers. To implement MTL for facial features, a pre-trained VGG-Face [10] model is used as a backbone for hair and face feature extraction. However, only the structure of the feature map extraction was maintained without modifications. Thus, the convolutional portion of the VGG-Face architecture is employed for feature extraction in our proposed architecture. This indicates that transfer learning was used to preserve the learning of the VGG-Face network for feature extraction. Meanwhile, an additional 13 dense layers are dedicated, i.e., our modification process, to support the realization of multitask learning and to implement 6 classification tasks. Moreover, two different datasets, i.e., FDDB and FRAED, were used to train the classification modules of the modified VGG-Face network architecture. Therefore, we trained six facial attributes in parallel using three classification modules, each of which is composed of two complementary tasks. Figure 7 illustrates the developed classification modules. These are module3-1 (hair-module), which performs hair color and hairstyle classification, module3-2 (gender-module), which detects facial smile detection and gender recognition, and module3-3 (faceID-module), which distinguishes face ID verification and age estimation.

The input to these modules is a feature map, extracted in the previous pipeline stage (module2) and fed as a flattening vector to the next stage (module3). When a classification module completes its task, the decision is sent to the edge server to complete the write-back stage (module4) for decision-making. At the server terminal, a list is used to gather decisions and write them into a CSV file as observations for all individuals detected in the input image during stage #1 (module1) using YOLOv8. It is important to note that we trained each module (hair, gender, and faceID) separately during the training phase, since the datasets are different. However, in the inference deployment, all three classification modules perform simultaneously. This indicates that when the feature maps of an inference from the previous stage (VGG-Face module) become available, all classifiers run to provide decisions, as shown in the lower portion of Figure 7. As observed, all the classification models require two fully connected layers for decision-making, except the face recognition model, which is constructed using three fully connected layers to increase the efficiency of face recognition. It is important to note that the ability to generalize the proposed edge computing-based architecture across various models primarily relies on two factors: the capability of the tasks to be learned simultaneously through the multi-task learning (MTL) approach, and the model's flexibility to be adapted for different tasks.

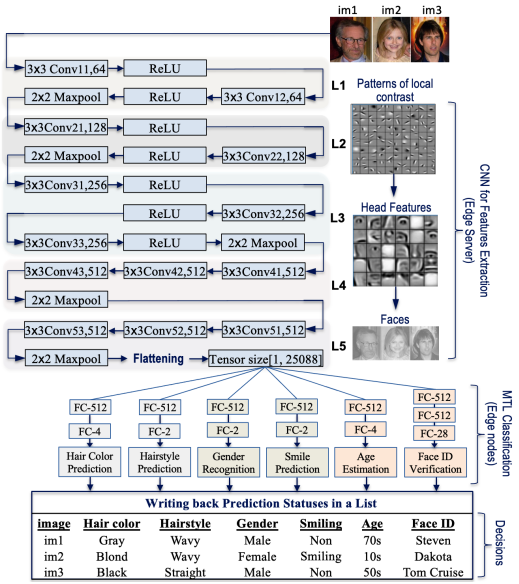

Figure 7. Modified VGG-Face network to support the multitask classification approach.

4.3. Computation Offloading of the Proposed Classification Modules

Here, we aim to implement edge intelligence, which combines AI and edge computing, in a pipeline-parallel embedded system to quickly analyze multiple individuals' faces. Edge computing has emerged recently to accelerate the execution of heavy computations for big data streaming applications. It involves leveraging multiple lightweight processing units at the edges of the cluster. Nevertheless, the majority of existing technologies send raw data to a remote server, which can be a burden for embedded systems equipped with resource-constrained edge devices. To address this limitation hindering the ability to distribute computational-intensive tasks on edge devices, computation offloading is used, through which intensive computations are assigned to the edge server to provide processing for raw data and extract feature maps. Meanwhile, lightweight computations, i.e., the classification modules, are offloaded to the edge devices. These modules receive the extracted feature maps from the server and perform corresponding classification tasks for seamless decision-making. Figure 8 depicts the offloading process of the proposed pipeline-parallel architecture described in Figure 6b. The cluster involves four computing devices, including a single powerful edge server and three Raspberry Pi edge computing devices. To enable the server to handle multiple clients' requests simultaneously, it is configured as a multithreaded server. Thus, the server assigns a thread for each edge device, and each thread works on two main temporary buffers (queues). It enqueues tensor elements to the feature map queue, whereas the received prediction outcomes for the edge nodes are inserted into the prediction queue. The two queues allocated to each edge device enable parallel processing for multiple classification tasks and parallel saving of prediction outcomes. Consequently, the deployed computation offloading approach refers to distributing face analysis computations based on task complexity and resource-rich/limited edge devices. In this context, extracted feature maps from detected heads are offloaded to the developed MTL-based classification modules, where these modules represent the dense layers of the modified VGG-Face model, illustrated in Figure 7. The modules are embedded in three edge devices and run a multithreading approach to perform three independent threads concurrently. Next, we discuss the levels of parallelism involved in the proposed pipeline-parallel architecture.

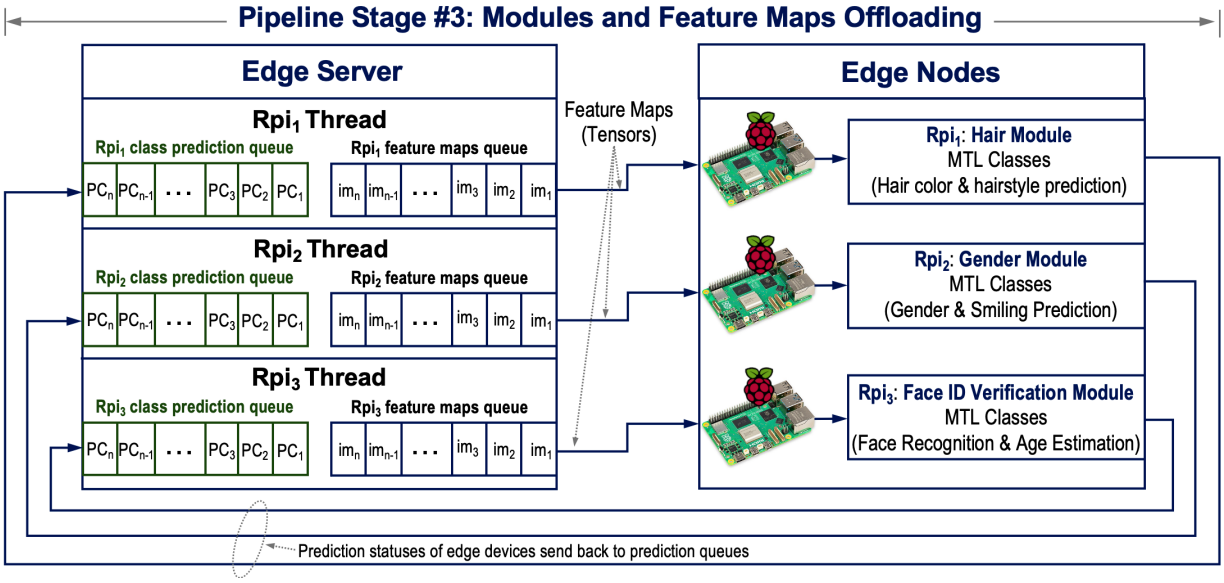

Figure 8. Offloading feature maps of detected heads to edge devices using multithreading.

4.4. Levels of Parallelism Leveraged in the Proposed Parallel Architecture

In computing paradigms, the levels of parallelism mainly include instruction-level parallelism (ILP), data-level parallelism (DLP), and thread-level parallelism (TLP). These levels of parallelism have been efficiently exploited in the design of the proposed pipeline-parallel architecture. For example, multiplying the kernel of a CNN filter with the same size image window, i.e., a 2-dimensional (2D) array taken from the input image, is achieved via the dot product that performs ILP and DLP on the filter coefficients and elements of the selected 2D array. It produces partial products, which are then summed to provide the feature maps. This natural parallelism is carried out intra-processor (within a processor core), which indicates that ILP and DLP can be very limited to exploiting only six cores of our edge server or three similar operations performed on an edge device equipped with a quad-core processor. Consequently, it is favorable to incorporate both ILP and DLP along with TLP for ultra-fast processing. This can be achieved by using pipeline and parallelism techniques among the processors to enable cooperation between intra- and inter-processor cores, resulting in high-speed performance for DL-based complex applications. However, to enable parallelism within dependent tasks, overlapping between them is required. This has been achieved via a hybrid approach that combines pipelining and producer–consumer techniques, as discussed in Section 4.

On the other hand, multithreading allows multiple threads to share processing units, which can be either among dependent tasks run on a single processor with an overlapping approach or among multiple independent threads run across a cluster of edge computing devices. Multithreading utilizes TLP to shrink the execution time per task. Therefore, to achieve high-speed performance with TLP, multicore processors can collaborate to run multiple independent threads simultaneously [60,61]. Hence, we utilize multithreading across multiple cores of processors to increase parallelism, allowing for the concurrent execution of independent threads on multiple cores of multiple processors. Moreover, the proposed pipeline-parallel design employs a multi-instruction stream multi-data stream (MIMD) architecture. This is due to multiple processing components executing various instruction streams and multiple images are processed by different processors' cores simultaneously. In contrast, the single-instruction stream multi-data stream (SIMD) architecture is implemented within each processor core by executing the same instruction stream, i.e., multiplication or addition, on different data streams, since each image is passed through multiple pipeline processing stages. Implementing the MIMD architecture instead of the SIMD archi-

tecture for the proposed deep face analysis system is advantageous. This approach allows for the simultaneous processing of multiple distinct instructions on different data streams, making it well-suited for the diverse tasks involved in various stages of the pipeline. Our architecture utilizes a multitask pipeline-parallel strategy, where different stages of the pipeline operate concurrently on separate processors. Consequently, the MIMD architecture supports this design by enabling distinct operations, such as detection and feature extraction, to occur in parallel without the synchronization constraints that are inherent in SIMD systems. Consequently, the proposed architecture utilizes parallelism at three levels (instruction, data, and thread). Furthermore, parallelism occurs across the multicore processor of the edge server and different processors of the edge cluster, including cores for the edge server and edge computing devices. This implies that the processing mechanism of the proposed architecture has been accelerated by running multiple independent threads simultaneously. Each algorithm is forwarded to a specific core, where each core performs a single module of the proposed pipeline-parallel face analysis architecture. For instance, DNN models that involve computationally intensive operations are directed to one core of the edge server processor, i.e., YOLOv8 and VGG-Face models are assigned to run on two cores of the edge server. On the other hand, the classification modules encompass lightweight operations compared to the YOLOv8 or VGG-Face networks; thus, a single co-processor of an edge device is dedicated to each classification module, as illustrated in Figure 9. Therefore, it perfectly aligns with the computationally intensive and task-diverse nature of the proposed edge computing-based framework.

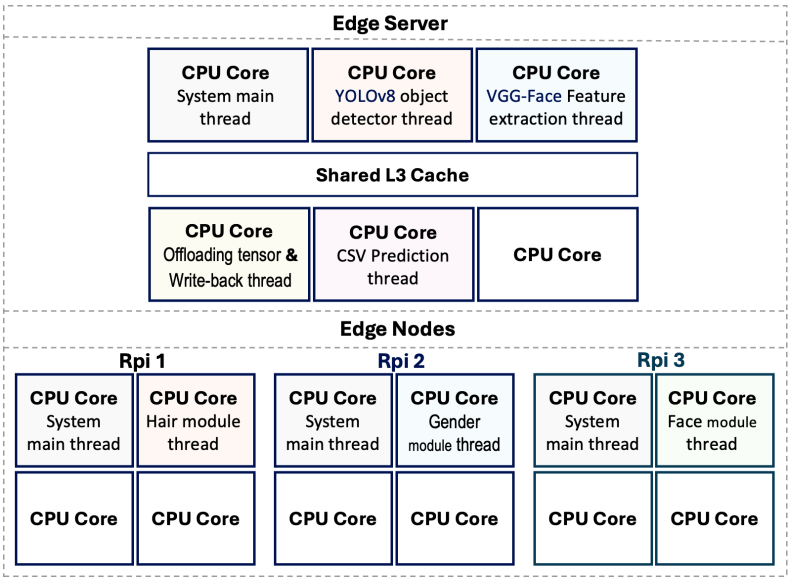

Figure 9. Multithreading of parallel modules on edge server and edge node processors.

5. Implementation and System Deployment

This section briefly introduces the required hardware and software tools for training, configuring, and implementing the proposed deep face analysis architecture. System deployment is also discussed at the end of this section.

5.1. Hardware

The testing cluster, in which tests have been deployed and executed, is a bare-metal cluster made up of four nodes: one master and three worker nodes, each one deployed in a different physical machine. The master is deployed on a laptop computer equipped with an Intel core i5 Processor, 16 GB of RAM, 1 TB disk capacity, and an NVIDIA GeForce GTX 1050 graphics card. The cluster of edge devices consists of three Raspberry Pi 4 Model

B single-board computers. Each Raspberry Pi has the following characteristics: 64-bit quad-core ARM Cortex-A72 processor, 8 GB of LPDDR4 RAM. A Hikvision Ethernet switch with five ports was used to connect the edge devices with the edge server through an Ethernet cable connection.

5.2. Software

Different software environments and tools were utilized to set up and configure these experiments. *VGG Image Annotator (VIA)* was used to manually generate the annotations of 2706 images selected from the FDDB dataset to create our HDDB dataset, which was essential to train the YOLOv8 head detection model. Likewise, the Shotwell program, available in Ubuntu OS, was used to manually crop 1400 head images and create the developed face recognition and age estimation dataset. Table 2 summarizes the utilized software and third-party library specifications. Two main operation systems were used: Ubuntu ran on the edge server, while Debian ran on the edge devices. Python versions 3.11.4 and 3.11.2 were installed on the server and edge computing devices, respectively. Moreover, Anaconda Navigator was used for smooth package management and launch. Among the various built-in IDEs provided by Anaconda Navigator, a Jupyter notebook was used as an integrated development environment (IDE) for training and building our deep learning MTL classifiers and YOLOv8 head detection model. To support faster parallel training on GPU, high-level feature tensor computation was leveraged with the PyTorch package through the torch.cuda library, thereby providing strong GPU acceleration. Thus, our developed multitask learning modules were implemented using the PyTorch deep learning framework to accelerate the models' training on a GPU. Note that the MTL-based classifiers were trained and constructed on the edge server. Then, they were deployed onto the edge devices for the multithreading inference architecture. The deployment process involved exporting them with the standard PyTorch extension, denoted by the (.pth file) extension. Figure 10 illustrates the practical deployment of the proposed face analysis framework architecture. As observed, the classification modules are offloaded on three resource-constrained edge computing devices. These edge devices work independently to provide parallel classification outcomes of multiple complementary tasks.

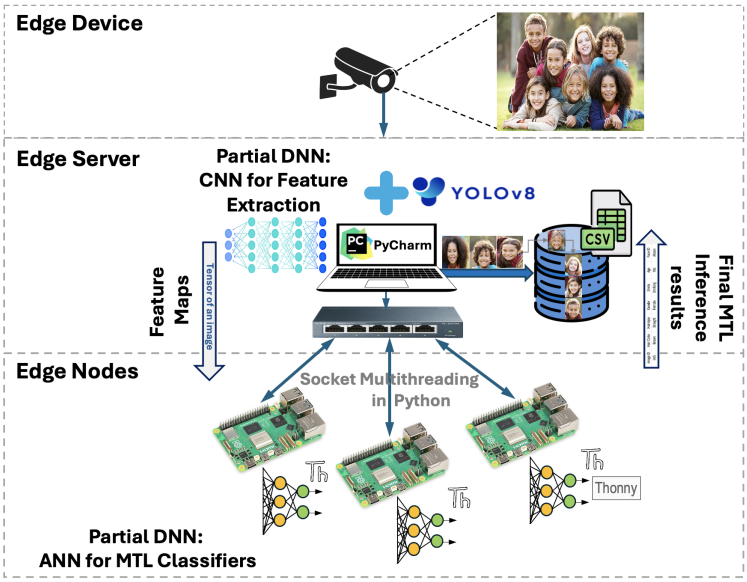

Figure 10. The framework of system deployment for the proposed deep face analysis.

Table 2. Software and third-party required libraries for training and deploying the proposed deep face analysis architecture.

Environment/Software	Version	Function
Ubuntu	23.10	Edge server OS
Debian	12 (bookworm)	Edge nodes OS
Edge Server Python	3.11.4	Development language
Edge Nodes Python	3.11.2	Development language
VGG Image Annotator	2	Manual annotation software
Shotwell	2.1	Cropping heads
Anaconda Navigator	2.4.2	Management tool
Jupyter Notebook	6.5.4	Integrated DL development environment
CUDA version	12.2	Edge server training DL models computing platform
Yolo	Ultralytics YOLOv8.0.194	Head detection algorithm
Server PyTorch version	2.1.0+cu121	DL framework
Nodes PyTortch version	2.1.1	DL framework
PyCharm version	17.0.10+1-b1087.17 AMD64	Server system development environment
Thonny version	4.1.4	Nodes system development environment
Socket	General socket API	Performing offloading operations

In the offloading process for system deployment, we depend on using a leader/follower architecture, where Python was utilized as the programming language for building the programs on both edges. On the server edge, PyCharm was used as the integrated IDE, whereas Thonny, a lightweight and resource-efficient IDE, was utilized on the edge nodes side. For communication between the edge server and edge devices, we employed Python's socket library and multithreading, through which efficient utilization and management of system resources were achieved. The edge server depends on using two main types of sockets working in different threads. The server socket thread listens for incoming connections from three Raspberry Pi devices via a transmission control protocol (TCP) socket, ensuring reliable and error-checked data transfer. Meanwhile, the client socket threads are responsible for managing the connections with edge nodes. For instance, when a client connects, i.e., Rpi1, the multithreaded server creates a new thread to handle server–client communication. Upon establishing, the thread becomes responsible for consuming feature tensors from the feature maps queue and pushing received prediction outcomes into the prediction queue, assigned for the corresponding connected edge device (Rpi1). Thus, employing multithreading at the server edge enables it to handle multiple clients' requests simultaneously without blocking. Similarly, socket threading is run on edge devices, where each device receives the tensor of the feature map from the server and passes it to the embedded MTL classification module for decision-making. After that, the predicted outcome is sent back to the server to be written in a CSV file, as illustrated in Figure 10. Next, we analyze and discuss the obtained results in terms of the prediction accuracy of the classification modules and the speedup of the proposed pipeline-parallel face recognition architecture compared to the sequential implementation.

6. Results and Discussion

The discussion and evaluation of the results are divided into three parts. In the first portion, we evaluate the detection performance of the YOLOv8 algorithm for head detection in terms of mean average precision (mAP) and inference time. Thus, the training of the YOLOv8 nano-version for face and head detection is evaluated on the edge server. In the

second part, accuracy, precision, recall, F1 score, and required training time are evaluated to validate the efficacy of the proposed classification modules. Therefore, we mainly concentrate on discussing and highlighting the results of the training phase, which was conducted on the server edge by running the Python CUDA package on the server's GPU. In the deployment phase, we evaluate the proposed pipeline-parallel architecture in terms of inference time and speedup. We deployed four computing architectures, two sequential (baseline) implementations with STL and MTL classification modules, respectively, and our two proposed pipeline-parallel architectures. In this context, the inference is implemented on both the edge server and edge computing devices.

6.1. Accuracy Evaluation of YOLOv8 Model

To evaluate the prediction accuracy of YOLOv8, we trained the model using the created HDDB dataset, which was specifically modified and tailored to train the YOLOv8 model for head detection. The dataset was divided into 70% training, 15% validation, and 15% testing (1894, 406, and 406, respectively). Precision, recall, and mAP are the most common object detection and performance evaluation metrics, as they comprehensively analyze the model's performance. Figure 11 validates the training processing of the YOLOv8 model. The model achieves a max mAP of 98.7% at epoch number 86 and an average loss of 1.2 for 100 epochs. Likewise, the mAP is around 75% when increasing the threshold for intersection over union (IOU) with the ground truth to become between 50 and 95. Considering the confusion matrix, the YOLOv8 model achieves high head detection outcomes, detecting 693 true-positive, 23 true-negative, and 47 false-negative in 608 images, as seen in Figure 12a. Moreover, Figure 12 shows YOLOv8 testing performance in terms of precision, recall, and the F1 score confidence curve. The model realizes precision-recall of 98.5% of all classes with mAP set up with a threshold level of 0.5, while the F1 score reaches 96% of all classes at a confidence of 0.662.

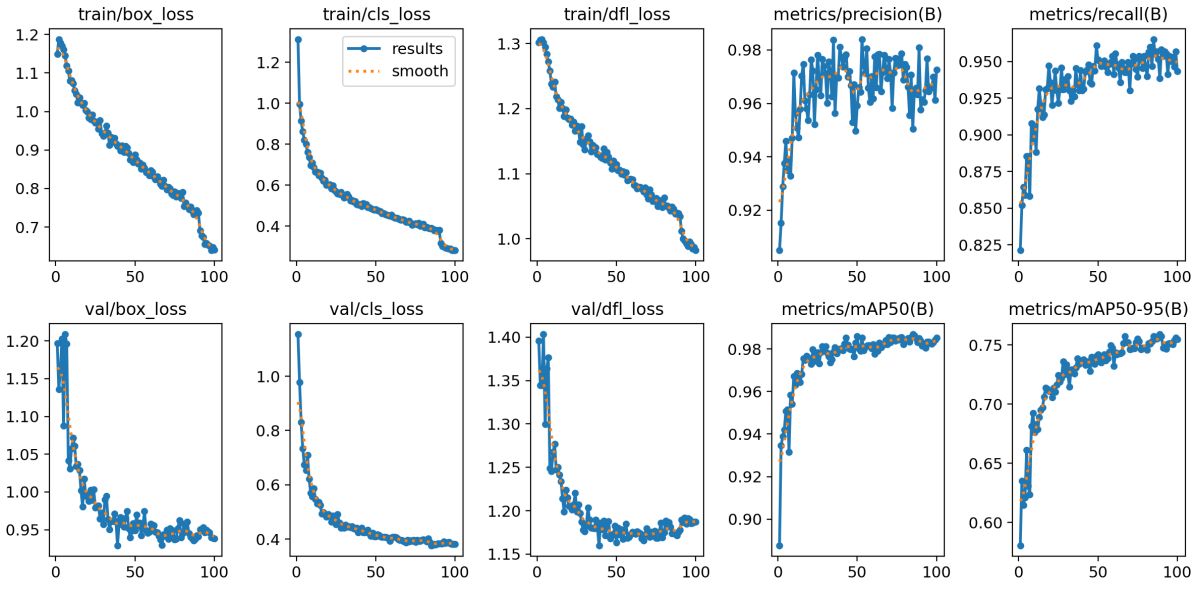

Figure 11. Training and validation performance of the YOLOv8 model for head detection. The x-axis represents the number of epochs.

On the other hand, the processing speed, such as the frame latency, is another valuable metric to ensure timely results for real-time applications. When deploying the proposed system and offloading the YOLOv8 algorithm to the edge server, with the specifications illustrated in Section 5.1, it takes roughly 34 ms to detect existing heads in the input image. Figure 13 illustrates sample results for head detection from the HDDB dataset. It further

demonstrates the validity of the model to accurately detect existing individuals' heads in an image.

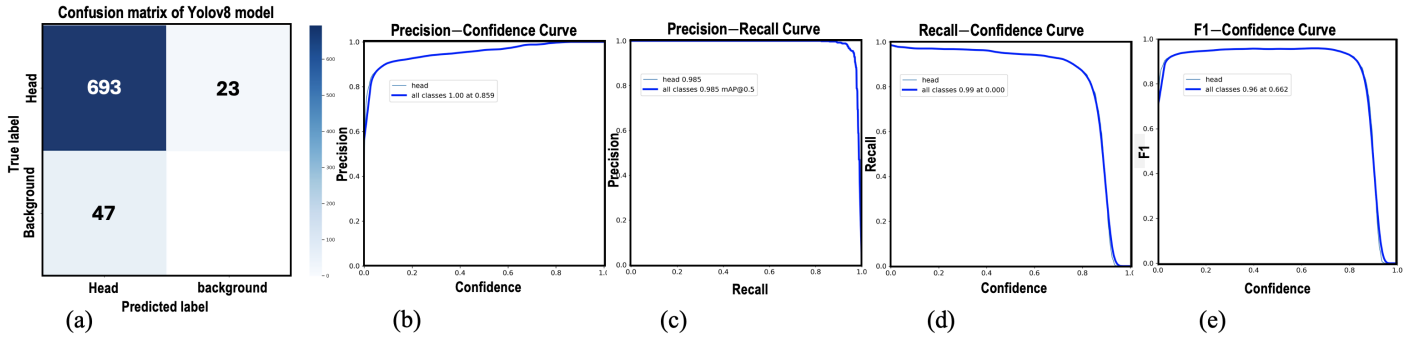

Figure 12. YOLOv8 testing performance; (**a**) confusion matrix, (**b**) precision, (**c**) recall, (**d**) precision-recall, and (**e**) F1 score confidence curve.

Figure 13. Head detection result samples of YOLOv8, where a red box denotes a detected head with its corresponding confidence level.

6.2. Accuracy Evaluation of Proposed Classification Modules

Here, the hair module, gender module, and face ID verification module were trained using two datasets (HDDB and FRAED), each divided into three subsets (train [80%], validation [10%], and test [10%]). To train the classification modules, we utilized the Stochastic Gradient Descent (SGD) optimizer with a momentum of 0.9, a learning rate of 0.002, and a batch size of 8. For the loss function, we used cross-entropy, which is well-suited for classification tasks as it emphasizes error distance and motivates the model to provide confident predictions. However, the Adam optimizer was found to work better for the face recognition and age estimation module; thus, we used the Adam optimizer for our third proposed classification module. Furthermore, to prevent interference while learning multiple tasks simultaneously, we first balanced the classes within the dataset to reduce the risk of overfitting to dominant classes. This balancing helps to enhance convergence across all tasks. Additionally, we optimized both the learning rate and batch size to further improve convergence across tasks. Finally, we monitored the performance of each task such that if a task experienced a delay in progress, its loss was temporarily increased to emphasize its importance during training.

The proposed MTL hair module provides competitive prediction performance compared to STL, as observed in Figure 14b,d. The MTL for color prediction slightly improved the prediction for black hair while misclassifying the brown class. Meanwhile, it provided almost the same color prediction performance for the hair modules of STL and MTL (88.4% and 88.55%, respectively), while the accuracy of the hairstyle was 88.55% in both learning approaches. In terms of training time, the STL hair color and hairstyle module incurred (4.37 and 4.33 ms) and (1.75 and 1.83 h) for epoch time and overall training time, respectively. However, the MTL reduced the training time almost by 50% since both features of hair color and style were learned concurrently. Furthermore, the MTL hair module achieved comparable outcomes for precession, recall, and F1 score compared to those of STL, as listed in Table 3. On the other hand, the proposed MTL gender-smile module achieved comparable prediction outcomes to the STL module. Figure 14f,h depict the confusion matrix of the module; as seen, the true and false negatives of both gender and smile were nearly identical to that in the STL-based module. This confirms the validation of the MTL training. In terms of training time, the MTL module of gender-smile required 4.39 ms for each epoch, whereas the STL modules required 4.37 and 4.33 ms for gender and smile, respectively. The evaluation of the module based on other metrics, such as precision, recall, and F1 score, demonstrates competitive prediction performance for the gender-smile module.

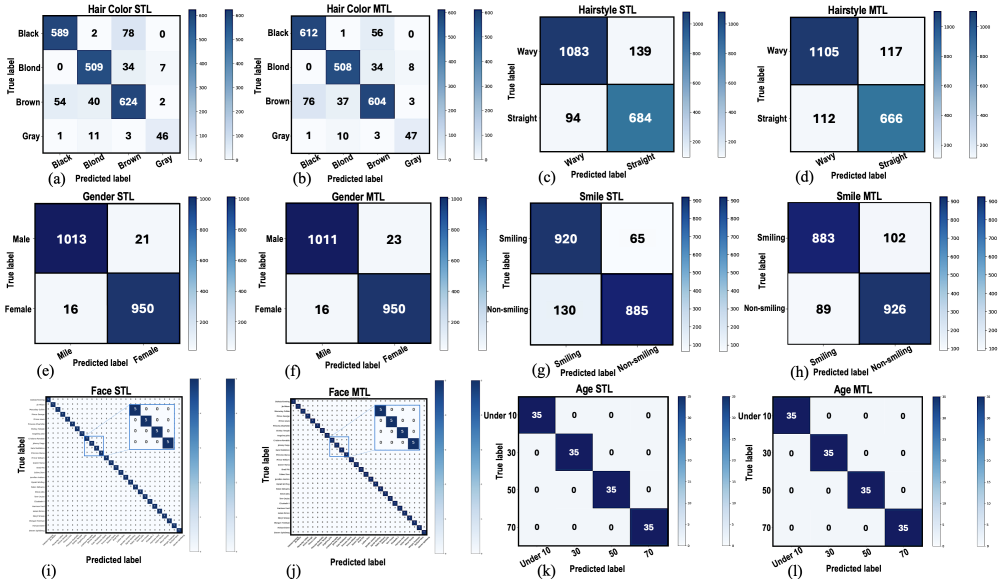

Figure 14. Confusion matrices for classification modules using STL and MTL; (**a**) hair color STL, (**b**) hair color MTL, (**c**) hairstyle STL, (**d**) hairstyle MTL, (**e**) gender STL, (**f**) gender MTL, (**g**) smile STL, (**h**) smile MTL, (**i**) Face STL, (**j**) Face MTL, (**k**) age STL, and (**l**) age MTL module.

Moreover, the FaceID module, which provides face recognition and age estimation, was evaluated based on the created face recognition and age estimation dataset (FRAED). As discussed in Section 3.4, each individual is represented by 50 images representing 4 life-age intervals, used for training, validation, and testing. Figure 14j,l depict the confusion matrices of face recognition and age estimation, respectively. The outcomes of both the STL and MTL modules show impressive prediction performances of 100%. Meanwhile, the proposed MTL module for face recognition delivered reduced training time; an improvement of 24.84% was realized. Considering the training time for age estimation, the proposed MTL module incurred a slightly higher training time, less than 1% (0.73%), as presented in the last three columns in row 2 of Table 3.

Table 3. Comparison of training the proposed classification modules using MTL versus STL. Note that the number of epochs is set to 20 for all conducted experiments.

Metric	Module 1				Module 2				Module 3			
	STL (Hair)		MTL (Hair)		STL		MTL		STL		MTL	
	Color	Style	Color	Style	Gender	Smile	Gender	Smile	Face	Age	Face	Age
Epoch Time (mins)	4.37	4.33	4.4		4.37	4.33	4.39		38	21	22	
Training Time (h)	1.75	1.83	1.75		1.78	1.75	1.77		10.95	8.17	8.23	
Accuracy (%)	88.4	88.35	88.55	88.55	98.15	90.25	98.05	90.45	100	100	100	100
Recall (%)	88.4	88.35	88.55	88.55	98.15	90.25	98.05	90.45	100	100	100	100
Precision (%)	88.45	88.55	88.5	88.56	98.15	90.43	98.05	90.45	100	100	100	100
F1 Score (%)	88.4	88.4	88.51	88.55	98.15	90.24	98.05	90.44	100	100	100	100

6.3. Speedup Evaluation of Proposed Pipeline-Parallel Architecture

The inference time of each algorithm in the proposed pipeline stages is computed to quantitatively evaluate the speed performance of the proposed deep face analysis architecture. As listed in Table 4, the pipeline-parallel architecture provides an improved execution time compared to the baseline sequential architecture with MTL, achieving a speedup of 23.95% when offloading the proposed pipeline-parallel architecture on a single-edge device. Despite the fact that the pure SIMD architecture incurs higher energy consumption due to low TLP exploitation [30], the proposed pipeline-parallel architecture increases processing efficacy by utilizing both DLP and TLP in a single edge device, running SIMD within each core and MIMD across the processor's cores. Therefore, a speedup of 44.88% was realized by distributing the classification modules on three edge computing devices. This was due to our proposed architecture utilizing data parallelism through multilane SIMD execution and thread parallelism through fully symmetric MIMD execution, enabling parallel execution of multiple threads running on multiple cores of microprocessors. Thus, offloading the classification tasks on a single edge device incurs a higher execution time (by 3.12 s) than deploying them on a cluster of three edge devices working in parallel. However, this speedup was achieved at the expense of deploying three edge computers instead of one edge quad-core computer. For instance, when the multithreading process of the classifiers is offloaded on a single edge device, the multithreading is performed on all four cores, where a single core is allocated for the software organization, and the other three cores are assigned for performing classification tasks. Each classification thread runs on a single core. Therefore, the speedup improvement factor for designs utilizing parallelism and multithreading is a tradeoff between the available edge devices in the deployed cluster and the execution time of the classification tasks.

To further demonstrate the efficacy of the proposed pipeline-parallel architecture, we evaluated its performance by executing a sequence of images ranging from 10 to 300, as illustrated in Figure 15. Offloading the classification modules on three edge devices resulted in the lowest execution time, as shown in Figure 15a. In terms of speedup, the proposed

pipeline-parallel architecture provided enhanced speedup when increasing the number of offloaded images. However, offloading the images on a single edge device delivered improved speed performance up to 120 images, whereas offloading the classification modules on three edge computing devices provided improved speedup even when reaching 300 images, as shown in Figure 15b. On average, a speedup of 32.61% and 62.48% was realized when distributing the classification modules on a cluster characterized by single-edge and three-edge devices, respectively, compared to the sequential implementation with MTL. Likewise, an average speedup of 44.4% could be achieved when offloading the classification modules onto three-edge computing devices compared to a single-edge device.

Table 4. Execution time of deploying the proposed deep face analysis architectures.

Architecture	Exec. Time of YOLOv8 (s)	Exec. Time of VGGFace (s)	Exec. Time of Classification (s)	Exec. Time of Write-Back (ms)	Overall Infer. Time (s)
Sequential implementation with STL	0.849	0.111	20.384	27	21.346
Sequential implementation with MTL	0.857	0.112	13.931	33	14.901
Pipeline-parallel with multithreading on 1 edge	1.102	0.128	10.102	0.2	11.332
Pipeline-parallel with multithreading on 3 edges	1.197	0.156	6.86	0.4	8.214

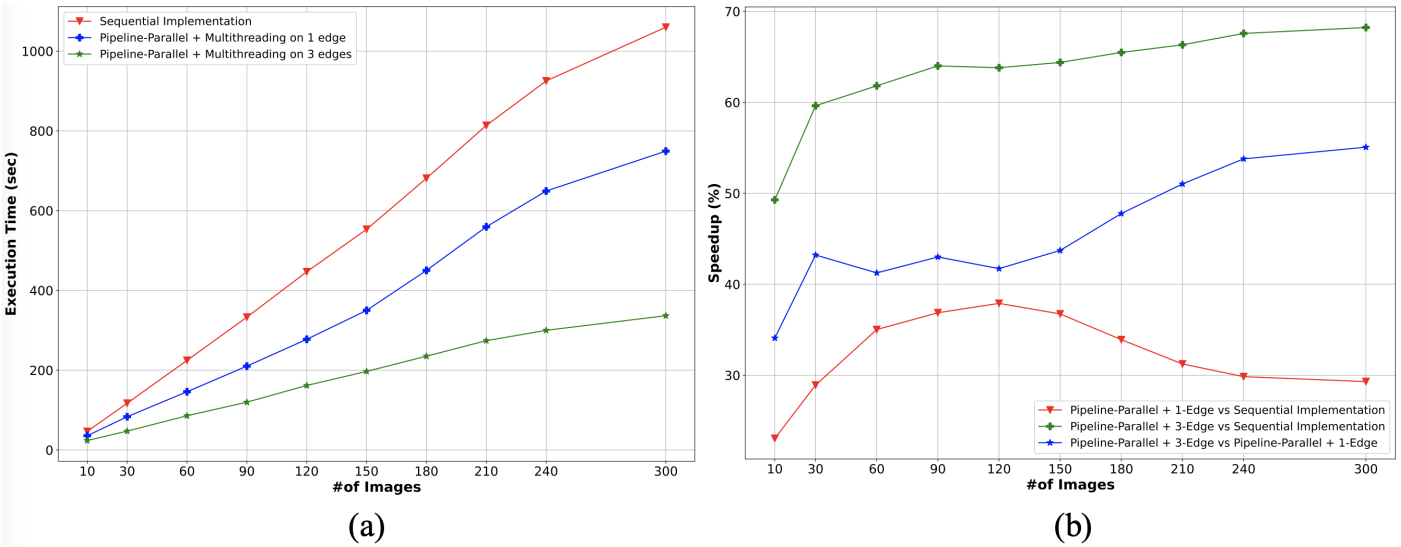

Figure 15. Speed performance evaluation of the proposed pipeline-parallel architecture; (**a**) execution time for pipeline-parallel configurations versus sequential implementation, (**b**) speedup comparisons of implemented configurations.

Additionally, we deployed the proposed pipeline-parallel architecture only on the server edge to further examine its speed acceleration efficacy using only the server edge. We evaluated the speedup obtained by executing a sequence of images ranging from 30 to 5000. Table 5 lists the execution time of each pipeline stage and the execution time of the sequential implementation with STL. As seen in the last column, an average speedup of

25.96% is achieved when implementing the innovative pipeline-parallel architecture only on the server edge.

Table 5. Speedup comparisons of the proposed deep face analysis architectures implemented on the server edge.

Architecture	# of Images	Exec. Time of YOLOv8 (s)	Exec. Time of VGGFace (s)	Exec. Time of Classifi. (s)	Write-Back Time (ms)	Overall Exec. Time (s)	Speedup (%)
Sequential + STL	1	0.88	0.181	0.014	0.00016	2.06	30.79
Pipeline-Parallel + MTL		1.257	0.136	0.012	0.00001	1.426	
Sequential + STL	30	5.519	3.099	0.381	0.00021	9.96	22.94
Pipeline-Parallel + MTL		7.526	6.248	6.065	6.054	7.675	
Sequential + STL	90	14.535	9.966	1.132	0.00036	26.59	27.74
Pipeline-Parallel + MTL		19.065	17.773	17.615	17.603	19.214	
Sequential + STL	120	19.127	12.321	1.527	0.00041	33.924	24.86
Pipeline-Parallel + MTL		25.34	24.125	23.95	23.939	25.489	
Sequential + STL	300	46.826	31.677	31.771	0.00093	83.184	26.19
Pipeline-Parallel + MTL		61.247	60.002	59.821	59.811	61.396	
Sequential + STL	1200	189.193	131.577	15.686	0.00312	337.371	26.24
Pipeline-Parallel + MTL		248.688	247.428	247.249	247.234	248.837	
Sequential + STL	5000	778.931	576.306	65.897	0.01379	1422.268	22.97
Pipeline-Parallel + MTL		1095.446	1094.142	1093.964	1093.95	1095.594	

In summary, the deep facial analysis architecture developed in this manuscript is an MTL-based CNN network designed to first detect heads and then analyze facial attributes such as age, gender, smiling, hairstyle and color, and face recognition simultaneously. It offers many advantages over dedicating a single classification module for each task, including improved computing efficiency, memory usage, and computational costs by deploying classification modules on resource-limited edge computers. Hence, the proposed framework has the potential to be used in various applications. One potential application is in forensic science, where it can assist in monitoring and identifying individuals who are targeting criminal areas. Another potential application is in the security of smart buildings, where it can detect and prevent unauthorized access when a suspicious person is identified.

7. Conclusions

This manuscript proposes a computation offloading architecture for deep face analysis that orchestrates cooperative DNN inference over a cluster of heterogeneous edge devices. An innovative framework architecture that utilizes pipeline and parallelism techniques for deep facial analytics is developed. The proposed pipeline-parallel architecture is designed to first detect heads and analyze various facial attributes such as hairstyle and color, age, gender, smile, and face identity verification simultaneously. Furthermore, the architecture employs an MTL-based CNN network to significantly shrink the structure of neural networks while maintaining high-quality classification outcomes. Additionally, multithreading is leveraged to distribute multiple tasks on multiple cores of the processor for parallel processing, considerably reducing the inference time required to classify and recognize facial features. Thus, the architecture is designed to provide detailed facial feature analysis of multiple individuals concurrently, thereby achieving an extended classification approach and high face recognition accuracy. The testing accuracy of the proposed classification modules ranges from 88.55% to 100%. Extensive experiments demonstrate the ability of the proposed pipeline-parallel architecture to learn and execute multiple dependent

and independent tasks concurrently, leading to a reduction in training and inference time by 50% and 44.88%, respectively. In terms of speedup, the proposed pipeline-parallel architecture, which offloads the classification modules on a cluster of three edge devices, achieves significant performance improvements (62.48% average speedup) compared to the sequential implementation of the deep face analysis model while deploying the classifiers on resource-constrained edge computing devices. Considering these favorable attributes, the proposed face analysis architecture can potentially be embedded in low-cost edge devices to capture images from surveillance cameras and provide real-time deep face analysis, making it suitable for deployment in areas such as security and forensic intelligence.

Author Contributions: Conceptualization, F.S.A. and B.T.H.; methodology, F.S.A.; software, B.T.H.; validation, F.S.A. and B.T.H.; formal analysis, F.S.A.; investigation, F.S.A.; resources, B.T.H.; data curation, B.T.H.; writing—original draft preparation, B.T.H.; writing—review and editing, F.S.A.; visualization, B.T.H.; supervision, F.S.A.; project administration, F.S.A.; funding acquisition, F.S.A. and B.T.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Ethical review and approval were waived for this study, as it did not involve humans or animals.

Informed Consent Statement: Not applicable, as the images used are from a publicly available dataset.

Data Availability Statement: The datasets used in the reported results can be requested from the corresponding author via email, as they are currently being utilized for ongoing research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhang, Z.; Luo, P.; Loy, C.C.; Tang, X. Facial Landmark Detection by Deep Multi-task Learning. In Proceedings of the Computer Vision–ECCV, Zurich, Switzerland, 6–12 September 2014; Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer International Publishing: Cham, Switzerland, 2014.
2. Ranjan, R.; Patel, V.M.; Chellappa, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *41*, 121–135. [CrossRef]
3. Hasan, M.M.; Hossain, M.A.; Srizon, A.Y.; Sayeed, A.; Ahmed, M.; Haquek, M.R. Improving performance of a pre-trained resnet-50 based VGGFace recognition system by utilizing retraining as a heuristic step. In Proceedings of the 24th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 18–20 December 2021.
4. Siddiqi, M.H.; Khan, K.; Khan, R.U.; Alsirhani, A.T. Face image analysis using machine learning: A survey on recent trends andapplications. *Electronics* **2022**, *11*, 1210. [CrossRef]
5. Shakeel, C.S.; Khan, S.J. Machine learning (ML) techniquesas effective methods for evaluating hair and skin assessments: Asystematic review. *Proc. Inst. Mech. Eng. Part H J. Eng. Med.* **2024**, *238*, 132–148. [CrossRef] [PubMed]
6. Martínez-Díaz, Y.; Nicolás-Díaz, M.; Méndez-Vázquez, H.; Luevano, L.S.; Chang, L.; González-Mendoza, M.; Sucar, L.E. Benchmarking lightweight face architectures on specific face recognition scenarios. *Artif. Intell. Rev.* **2021**, *54*, 6201–6244. [CrossRef]
7. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2012; Volume 25.
8. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In Proceedings of the 22th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17), Xi'an, China, 8–12 April 2017; Association for Computing Machinery: New York, NY, USA, 2017.
9. Mishra, R.K.; Reddy, G.S.; Pathak, H. The understanding of deep learning: A comprehensive review. *Math. Probl. Eng.* **2021**, *2021*, 5548884. [CrossRef]
10. Parkhi, O.; Vedaldi, A.; Zisserman, A. Deep face recognition. In Proceedings of the British Machine Vision Conference (BMVC), Swansea, UK, 7–10 September 2015; pp. 1–12.

11. Taigman, Y.; Yang, M.; Ranzato, M.; Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014; pp. 1701–1708.
12. Schroff, F.; Kalenichenko, D.; Philbin, J. Facenet: A unified embedding for face recognition and clustering. In Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 815–823.
13. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
14. Li, S.; Zhou, Z.; Zhao, M.; Yang, J.; Guo, W.; Lv, Y.; Kou, L.; Wang, H.; Gu, Y. A multitask benchmark dataset for satellite video: Object detection, tracking, and segmentation. *IEEE Trans. Geosci. Remote Sens.* **2023**, *61*, 1–21. [CrossRef]
15. Aggarwal, A.; Kumar, V.; Gupta, R. Object detection based approachesin image classification: A brief overview. In Proceedings of the 2023 IEEE Guwahati Subsection Conference (GCON), Guwahati, India, 23–25 June 2023.
16. GitHub-Ultralytics. Available online: https://github.com/ultralytics/ultralytics/releases/tag/v8.1.0 (accessed on 23 November 2024).
17. Ryan, C.; Elrasad, A.; Shariff, W.; Lemley, J.; Kielty, P.; Hurney, P.; Corcoran, P. Real-time multi-task facial analytics with event cameras. *IEEE Access* **2023**, *11*, 76964–76976. [CrossRef]
18. Foggia, P.; Greco, A.; Saggese, A.; Vento, M. Multi-task learning on the edge for effective gender, age, ethnicity and emotion recognition. *Eng. Appl. Artif. Intell.* **2023**, *118*, 105651. [CrossRef]
19. Hu, G.; Liu, L.; Yuan, Y.; Yu, Z.; Hua, Y.; Zhang, Z.; Shen, F.; Shao, L.; Hospedales, T.; Robertson, N.; et al. Deep multi-task learning to recognise subtle facial expressions of mental states. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 103–119.
20. Yang, D.; Li, X.; Dia, X.; Zhang, R.; Qi, L.; Zhang, W.; Jiang, Z. All in one network for driver attention monitoring. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 2258–2262.
21. Li, W.; Mikailov, M.; Chen, W. Scaling the inference of digital pathology deep learning models using CPU-based high-performance computing. *IEEE Trans. Artif. Intell.* **2023**, *4*, 1691–1704. [CrossRef]
22. Zhou, H.; Li, M.; Wang, N.; Min, G.; Wu, J. Accelerating deep learning inference via model parallelism and partial computation offloading. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 475–488. [CrossRef]
23. Lee, S.H. Real-time edge computing on multi-processes and multithreading architectures for deep learning applications. *Microprocess. Microsystems* **2022**, *92*, 104554. [CrossRef]
24. Chen, X.; Zhang, J.; Lin, B.; Chen, Z.; Wolter, K.; Min, G. Energy efficient offloading for DNN-based smart IoT systems in cloud-edge environments. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 683–697. [CrossRef]
25. Hu, S.; Li, M.; Gao, J.; Zhou, C.; Shen, X. Adaptive device-edge collaboration on DNN inference in AIoT: A digital-twin-assisted approach. *IEEE Internet Things J.* **2024**, *11*, 12893–12908. [CrossRef]
26. Qawaqneh, Z.; Mallouh, A.A.; Barkana, B.D. Deep convolutional neural network for age estimation based on VGG-face model. *arXiv* **2017**, arXiv:1709.01664.
27. Daoud, E.A.; Samara, G. Improving the face recognition performance using Gabor and VGGFace2 features concatenation. In Proceedings of the 2022 6th International Conference on Information Technology (InCIT), Nonthaburi, Thailand, 10–11 November 2022; pp. 187–190.
28. Muhammad, U.R.; Svanera, M.; Leonardi, R.; Benini, S. Hair detection, segmentation, and hairstyle classification in the wild. *Image Vis. Comput.* **2018**, *71*, 25–37. [CrossRef]
29. Kristiani, E.; Yang, C.-T.; Huang, C.-Y. iSEC: An optimized deep learning model for image classification on edge computing. *IEEE Access* **2020**, *10*, 27267–27276. [CrossRef]
30. Cheikh, A.; Sordillo, S.; Mastrandrea, A.; Menichelli, F.; Scotti, G.; Olivieri, M. Klessydra-T: Designing vector coprocessors for multithreaded edge-computing cores. *IEEE Micro* **2021**, *41*, 64–71. [CrossRef]
31. Goel, A.; Tung, C.; Hu, X.; Thiruvathukal, G.K.; Davis, J.C.; Lu, Y.-H. Efficient computer vision on edge devices with pipeline-parallel hierarchical neural networks. In Proceedings of the 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), Taipei, Taiwan, 17–20 January 2022; pp. 532–537.
32. Hao, Z.; Xu, G.; Luo, Y.; Hu, H.; An J.; Mao, S. Multi-agent collaborative inference via DNN decoupling: Intermediate feature compression and edge learning. *IEEE Trans. Mob. Comput.* **2023**, *22*, 6041–6055. [CrossRef]
33. Gamatie, A.; Devic, G.; Sassatelli, G.; Bernabovi, S.; Naudin, P.; Chapman, M. Towards energy-efficient heterogeneous multicore architectures foredge computing. *IEEE Access* **2019**, *7*, 49474–49491. [CrossRef]
34. Gkioxari, G.; Hariharan, B.; Girshick, R.; Malik, J. R-CNNs for pose estimation and action detection. *arXiv* **2014**, arXiv:1406.5212.
35. Simonyan, K.; Zisserman, A. Very deep convolutional networks forlarge-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.

36. Cao, Q.; Shen, L.; Xie, W.; Parkhi, O.M.; Zisserman, A. VGGFace2: A dataset for recognising faces across pose and age. In Proceedings of the 2018 13th IEEE International Conference on Automatic Face and Gesture Recognition (FG2018), Xi'an, China, 15–18 May 2018; pp. 67–74.
37. Toseeb, U.; Keeble, D.R.; Bryant, E.J. The significance of hair forface recognition. *PLoS ONE* **2012**, *7*, e34144. [CrossRef] [PubMed]
38. Yacoob, Y.; Davis, L. Detection and analysis of hair. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 1164–1169. [CrossRef] [PubMed]
39. Airlie, M.; Robertson, J.; Ma, W.; Airlie, D.; Brooks, E. A novel application of deep learning to forensic hair analysis methodology. *Aust. J. Forensic Sci.* **2022**, *56*, 311–322. [CrossRef]
40. Borza, D.; Ileni, T.; Darabant, A. A deep learning approach to hair segmentation and color extraction from facial images. In Proceedings of the Advanced Concepts for Intelligent Vision Systems: 19th International Conference, ACIVS 2018, Poitiers, France, 24–27 September 2018; Springer International Publishing: Cham, Switzerland, 2018; pp. 438–449.
41. Kim, D.; Lee, E.; Yoo, D.; Lee, H. Fine-grained human hair segmentation using a text-to-image diffusion model. *IEEE Access* **2014**, *12*, 13912–13922. [CrossRef]
42. Khan, K.; Attique, M.; Syed, I.; Sarwar, G.; Irfan, M.A.; Khan, R.U. Aunified framework for head pose, age and gender classification through end-to-end face segmentation. *Entropy* **2019**, *21*, 647. [CrossRef]
43. Benini, S.; Khan, K.; Leonardi, R.; Mauro, M.; Migliorati, P. Face analysis through semantic face segmentation. *Signal Process. Image Commun.* **2019**, *74*, 21–31. [CrossRef]
44. Cherian, A.K.; Devipriya, S.; Saoji, B.P.; Mallikeswari, B.; Thiagarajan, R.; Krishnamoorthy, R. A robust design of real-time resilient smile recognition system using hybrid deep learning principles. In Proceedings of the 2024 10th International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 12–14 April 2014; pp. 592–596.
45. Ye, J.; Yu, Y.; Zheng, Y.; Liu, Y.; Wang, Q. Dep-FER: Facial expression recognition in depressed patients based on voluntary facial expression mimicry. *IEEE Trans. Affect. Comput.* **2024**, *15*, 1725–1738. [CrossRef]
46. Hosseinzadeh, M.; Wachal, A.; Khamfroush, H.; Lucani, D.E. Optimal accuracy-time trade-off for deep learning services in edge computing systems. In Proceedings of the ICC 2021—IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
47. Zhu, Y.; Wang, Z.; Han, Z.; Li, N.; Yang, S. Multithread optimal offloading strategy based on cloud and edge collaboration. In Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020—Spring), Antwerp, Belgium, 25–28 May 2020; pp. 1–5.
48. Li, J.; Liang, W.; Li, Y.; Xu, Z.; Jia, X.; Guo, S. Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism. *IEEE Trans. Mob. Comput.* **2023**, *22*, 3017–3030. [CrossRef]
49. Mao, J.; Chen, X.; Nixon, K.W.; Krieger, C.; Chen, Y. Modnn: Localdistributed mobile computing system for deep neural network. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1396–1401.
50. Zeng, L.; Chen, X.; Zhou, Z.; Yang, L.; Zhang, J. CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Trans. Netw.* **2021**, *29*, 595–608. [CrossRef]
51. Kokkinos, I. Ubernet: Training a universal convolutional neural networkfor low-, mid-, and high-level vision using diverse datasets and limited memory. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5454–5463.
52. Caruana, R. Learning to learn. In *Multitask Learning*; Springer: Boston, MA, USA, 1998; pp. 95–133.
53. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
54. Wang, Q.; Guo, G. Benchmarking deep learning techniquesfor face recognition. *J. Vis. Commun. Image Represent.* **2019**, *65*, 102663. [CrossRef]
55. Zhu, X.; Ramanan, D. Face detection, pose estimation, and landmark localization in the wild. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 2879–2886.
56. Dhamecha, T.I.; Singh, R.; Vatsa, M.; Kumar, A. Recognizing disguised faces: Human and machine evaluation. *PLoS ONE* **2014**, *19*, e99212. [CrossRef] [PubMed]
57. Jain, V.; Learned-Miller, E. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*; University of Massachusetts: Amherst, MA, USA, 2010.
58. Dutta, A.; Zisserman, A. The via annotation software for images, audioand video. In Proceedings of the 27th ACM International Conference on Multimedia (MM '19), Nice, France, 21–25 October 2019; pp. 2276–2279.
59. Liu, Z.; Luo, P.; Wang, X.; Tang, X. Deep learning face attributes in the wild. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 3730–3738.

60. Solihin, Y. *Fundamentals of Parallel Multicore Architecture*; CRC Press, Taylor and Francis Group: Boca Raton, FL, USA, 2016.
61. Hennessy, J.L.; Patterson, D.A. *Computer Architecture: A Quantitative Approach*, 6th ed.; Morgan Kaufmann of Elsevier; Elsevier: Amsterdam, The Netherlands, 2019.